PRZEMYSŁOWY INSTYTUT AUTOMATYKI I POMIARÓW
MERA-PIAP
Al. Jerozolimskie 202 02-222 Warszawa Telefon 23-70-81

Ośrodek Automatyzacji Procesów Produkcji

440 BE10

Główny wykonawca dr inż. Marian Wrzesień

Wykonawcy mgr inż. A. Bratek
mgr inż. Z. Stańczak

Konsultant

Nr zlecenia S 1300

"Adaptacja systemu czasu rzeczywistego OS-9 na bazie minikomputera przemysłowego PEP MC do celów zintegrowanej automatyzacji produkcji."

Etap 2 pt.: "Modyfikacja założeń techniczno-ekonomicznych dla modernizacji fabryki kół zębatych FZ-200 w Stalowej Woli SA z uwzględnieniem specyfiki komputera PEP MC".

Zleceniodawca praca w ramach działalności statutowej PIAP

Pracę rozpoczęto dnia czerwiec 1992 **zakończono dnia** 31.08.1992 r.

Kierownik Ośrodka

dr inż. M. Wrzesień

Z-ca Dyrektora d/s Badawczo-Rozwojowych

dr inż. J. Jabłkowski

Praca zawiera:

stron 43
rysunków –
fotografii –
tabel –
tablic –
załączników –

Nr rejestr. 6862

Rozdzielnik - ilość egz:

Egz. 1 BOINTE
Egz. 2 OAP
Egz. 3 OAP a/a
Egz. 4
Egz. 5
Egz. 6

Analiza deskryptorowa

INFORMATYKA
WD MASZYNY MATEMATYCZNE
WD KOMPUTERY

Analiza dokumentacyjna

Sprawozdanie zawiera analizę systemu OS-9 oraz modyfikację założeń techniczno-ekonomicznych.

Tytuły poprzednich sprawozdań

1. "Adaptacja systemu czasu rzeczywistego OS-9 na bazie minikomputera przemysłowego PEP MC do celów zintegrowanej automatyzacji produkcji."
   Etap 1 pt.: "Zakup komputera z firmy PEP MC w wersji development system (sprzęt + oprogramowanie)".

UKD

PIAP 41/88 10000

Spis treści

1. Wprowadzenie

Zakres niniejszego etapu pracy wynika z przewidywanej współpracy PIAP z hutą Stalowa Wola S.A. dotyczącej opracowania nowego systemu sterowania zautomatyzowaną linią do obróbki kół zębatych FZ-200, zapewniającego zmodernizowanie poziomu zarządzania produkcją. Prace wstępne w tym zakresie zostały zrealizowane na podstawie umowy (Nr. 240/90/u z dnia 1990.11.26) zawartej pomiędzy Przemysłowym Instytutem Automatyki i Pomiarów PIAP, a Hutą Stalowa Wola S.A., wg zlecenia HSW S.A. (Nr RA/381/90 z dnia 1990.11.09). Należy podkreślić, że aktualnie PIAP zabiega o nawiązanie współpracy z HSW S.A. przy modernizacji fabryki FZ-200, na zasadach realizacji projektu celowego wspieranego przez Komitet Badań Naukowych.

Celem podjętej wcześniej współpracy było wypracowanie materiałów decyzyjnych i wyjściowych, w formie Założeń Techniczno-Ekonomicznych, przed przystąpieniem do właściwej modernizacji sprzętu i oprogramowania systemu. ZTE są zarejestrowane w PIAP pod nr rej. 6636, i nie będą załączane do nn. sprawozdania. Będą natomiast stanowiły odniesienie przy pracach nn. etapu zlecenia S1300.

Modyfikacja ZTE, z względnieniem specyfiki minikomputera PEP MC, wymagała poznania takich jego elementów środowiska jak:

1. system operacyjny OS-9,
2. zestaw narzędzi programowania minikomputera,
3. system komunikowania się komputera PEP MC z otoczeniem.

Poniżej, w pkt. 2., 3., i 4., omówiono wyszczególnione zagadnienia. Punkt 5. stanowi podsumowanie istoty różnic pomiędzy opracowanymi wcześniej ZTE, a potrzebami wynikającymi z zastosowania minikomputera PEP MC.

2. System operacyjny OS-9 i środowisko OS-9 minikomputera PEP MC

2.1 Opis ogólny

2.1.1 Wstęp

System operacyjny jest nadrzędnym nadzorcą zasobów i funkcji systemu komputerowego. Zasoby komputera zawierają pamięć, urządzenia czasowe i wejścia/wyjścia CPU takie, jak terminale, napędy dysków i drukarki. Podstawowe funkcje systemu OS-9 to:

- pełnienie roli interfejsu pomiędzy komputerem, a użytkownikiem,
- zarządzanie operacjami wejście/wyjście (I/O) systemu,
- realizacja ładowania i wykonywania programów,
- tworzenie i zarządzanie systemem katalogów i plików,
- zarządzanie podziałem czasu i wielozadaniowością,
- przydział pamięci dla wszelkich celów działania systemu.

Pełne wykorzystanie systemu operacyjnego jest możliwe dzięki wykorzystaniu programów aplikacyjnych, nie stanowiących elementów systemu operacyjnego, takich jak procesora tekstowego, czy pakietów obliczeniowych. Ponadto, do systemu operacyjnego dołaczonych jest ponad 70 programów obsługi systemu, które stanowią małe programy aplikacyjne umożliwiające nadzór, zarządzanie, konserwację i modyfikację funkcji systemu.

Funkcje systemu OS-9 mogą być używane na dwa sposoby:

1. Poprzez program interpretera poleceń **shell**, oraz
2. Poprzez system przerwań, na którego bazie zostało opracowanych szereg programów obsługi systemu, jak ładowanie programu do pamięci, tworzenie nowych zadań, tworzenie i usuwanie plików, czytanie, pisanie do plików itp.

2.1.2 Wielozadaniowość i wielodostępność

Wielozadaniowość (multitasking) umożliwia realizację wielu zadań w tym samym czasie. Do tego celu jest wykorzystywane przełaczanie w czasie wykonywania każdego z zadań postawionych systemowi do wykonania w tym samym czasie. Występuje tu podział na zadania pierwszoplanowe (foreground) i zadania wykonywane w tle (background).

Wielodostępność (multiuser), lub inaczej podział czasu, (timesharing) jest naturalnym rozszerzeniem funkcji wielozadaniowości systemu. Wielodostępność zapewnia zabezpieczone sterowanie dostę-

pem z podziałem czasu przy jednoczesnym spełnianiu prywatności w ramach systemu.

Zakupiony przez PIAP zestaw PEP MC "development system" aktualnie umożliwia pracę współbieżną czterem użytkownikom systemu. Jednakże architektura tego systemu nie wprowadza ograniczeń na liczbę użytkowników.

2.1.3 Podzielny system modułowy

Podzielny system modułowy zapewnia jednokrotne załadowanie określonego programu do pamięci nawet przy wielokrotnym wywołaniu, przez kilku użytkowników, tego samego modułu programu. Ta cecha systemu powoduje oszczędne korzystanie z zasobów pamięci komputera. Podobnie realizowane jest korzystanie z modułów umieszczonych w bibliotece dołączonej do systemu.

2.2 Uruchamianie, zamykanie i dostęp do systemu

System OS-9 został przekazany do PIAP w formie zestawu dyskietek uruchomieniowych wraz z opisami systemu. Ze względu jednakże na to, że system ten został wgrany na dysk twardy (zgodnie z przyjętą praktyką dostawców), nie było konieczności formatowania dysku twardego i instalowania na nim systemu.

Pliki wchodzące w skład systemu, jak i pliki uzupełniające, są usystematyzowane w strukturze hierarchicznej (drzewowej). Pliki zawarte w poszczególnych katalogach i podkatalogach wymienione są w pliku tekstowym **/dd/release.doc**. Obecność dysku twardego powoduje, że start zimny (bootstraping) odbywa się automatycznie po załączeniu komputera, bez potrzeby stosowania dyskietek systemowych. Wystartowanie systemu jest równoznaczne z wywołaniem opcjonalnej części pakietu profesjonalnego OS-9, a mianowicie ROM-Debuggera. Ten niekonwencjonalny program może być wywoływany po starcie, jak i po zamknięciu systemu. W celu wprowadzenia systemu do pamięci należy podać z terminala o najwyższym priorytecie (interfejs szeregowy nr 1 pakietu VM20 komputera - /term) polecenie ROM-Debuggera: <os>.

W trakcie uruchamiania systemu są wczytywane moduły systemu OS-9, zawarte w pliku **/dd/OS9boot**, natomiast plik **/dd/startup**, ustanowiony przez użytkownika, zawiera zestaw poleceń procedur interpretera shell, dobrany wg aktualnych potrzeb stosującego system. W pliku tym zawarte są dyrektywy inicjalizacji poszczególnych urządzeń WE/WY systemu, dla których firma Microware przyjęła poniższe oznaczenia:

- TERM	Pierwotny terminal systemu
- t1, t2,....	Inne terminale szeregowe
- p	Drukarka równoległa
- p1	Drukarka szeregowe
- dd	Napęd dyskowy domyślny
- d0	Jednostka dysku elastycznego nr 0
- d1, d2,....	Inne dyski elastyczne
- h0, h1,....	Dyski twarde z zabronionym formatowaniem
- h0fmt, h1fmt,....	Twarde dyski
- n0, n1,....	Urządzenia sieciowe
- mt0, mt1,..	Pamięci taśmowe
-r0	RAM-dysk

Dostęp do systemu następuje po podaniu nazwy użytkownika, oraz podaniu hasła, które nie jest wyświetlane na monitorze przy jego wprowadzaniu z klawiatury. Nazwa użytkownika oraz hasło zawarte są w pliku tekstowym **/dd/sys/password.** Każdemu użytkownikowi odpowiada jedna linia tego pliku, której pola, oddzielone przecinkami, oznaczają:

1. Nazwa użytkownika (do 32 znaków ze spacjami włącznie. Brak nazwy powoduje dowolność wprowadzanej nazwy).
2. Hasło (do 32 znaków ze spacjami włącznie. Pominięcie tego pola zwalnia od wymagania podawania hasła).
3. Numer identyfikacyjny "grupa.użytkownik". (Tak grupa, jak i użytkownik mogą przyjmować wartości 0 - 65535. Każdy z użytkowników powinien mieć oddzielny numer identyfikacyjny).
4. Inicjowany priorytet procesu (przyjmuje wartość od 1 do 65535. Wskazuje priorytet wykonywania postawionego zadania).

5. Inicjowany katalog wykonawczy (domyślnie **/d0/CMDS**. Jeśli tak, to wystarczy w to pole wstawić (.), w przeciwnym razie należy podać ścieżkę).

6. Inicjowany katalog danych (podaje się ścieżkę, w przeciwnym razie (.) ustanawia się katalog bieżący jako początkowy katalog danych).

7. Inicjowany program (nazwa i parametry programu wykonwanego wstępnie, zwykle shell).

Przykład:

marian.september, 7.7,100,.,/dd/usr/marian,shell

Dostęp nadzorcy programu następuje po podaniu nazwy **super** oraz podaniu właściwego hasła.

Przykładowa linia w pliku **/dd/sys/password** dla nadzorcy ma postać:

super.user, 0.0,255,.,.,shell

Zamykanie systemu może być przeprowadzone:

1. poprzez odłączenie zasilania sieciowego,
2. poprzez podanie polecenia **break**, oraz
3. poprzez wykonanie procedury powodującej odłączenie kolejno: sieci, interfejsów WE/WY (spooler) oraz zamknięcie systemu instrukcją **break**. W zależności od potrzeb i konieczności (np. wykonanie określonych procedur przed zamknięciem systemu) należy wybrać jedną z powyższych metod.

2.3 Stosowanie klawiatury i monitora

Klawiatura i monitor umożliwiają komunikację z systemem za pomocą programu KERMIT, który jest licencjonowanym oprogramowaniem, w które dostawca wyposaża system. W zależności od typu stosowanego terminala komputera PEP MC, wybierany jest opis charakterystyki terminala z pliku **/dd/SYS/termcap**. Szczegółowy zestaw terminali zawarto w pliku **/dd/sys/termset**. Pliki te umożliwiają wzajemne dopasowanie do siebie wybranego terminala i systemu. W przypadku zastosowania komputera PC 386/25 jako terminala, przyjmuje się typ terminala TV 920. Praca terminala w systemie Windows natomiast, wymaga przyjęcia typu vt 100.

O ile wybór typu terminala nie jest wyraźnie wskazany podczas pracy nadzorczej systemu, to praca z edytorem uMACS, stanowiącym element uzupełniający systemu, narzuca ten wybór w sposób krytyczny.

Podstawowe funkcje edytora poleceń podawanych z klawiatury to (^ ≡ Ctrl):

- ^A Powtarza poprzedniego polecenia
- ^D Powtarza wyświetlenie linii bieżącej
- ^H Kasuje poprzedni znak
- ^Q Podejmuje uprzednio wstrzymane działania przez ^S
- ^S Wstrzymuje działanie WE i WY, aż do podania ^Q
- ^W Czasowo wstrzymuje przewijanie ekranu
- ^X Usuwa linię
- ^[Wskazuje koniec pliku
- ^C Wysyła sygnał przerwania dla nalpóźniejszego programu
- ^E Wysyła przerwanie aktualnie wykonywanego programu

2.4 Nadawanie nazw plikom

Nazwy plików mogą się składać ze znaków: liter (A - Z), liter (a - z), cyfr (0 - 9), znaku podkreślenia (_), kropki (.) i znaku dolara ($) tak, aby łącznie ich liczba nie przekraczała 28 pozycji, przy czym nazwa musi zawierać co najmniej jedną literę lub cyfrę i nie może zawierać spacji. Pliki rozpoczynające się kropką są plikami ukrytymi (np. .login .logout czy umacsrc). Stosuje się znaki zastępcze * oraz ? o funkcjach zgodnych ze znanymi funkcjami MS DOS.

2.5 Funkcja shell

Funkcja shell jest interpreterem poleceń systemu OS-9. Pozwala także ustanawiać środowisko konfigurowane przez użytkownika. Po uruchomieniu systemu, i wykonaniu programu inicjującego określonego w pliku /dd/sys/password, rozpoczyna się program shell, którego działanie jest potwierdzane wyświetlaniem ponaglenia (prompt). Od

tego momentu przyjmowane są polecenia systemu oraz opcje programu shell, a wśród nich:

-e=<file> powoduje drukowanie komunikatów błędów w oparciu o plik /dd/sys/errmsg.

-ne wstrzymuje wyświetlanie komunikatów o błędach (domyślna)

-l żąda polecenia <logout> przy zaprzestaniu *login* funkcji *shell*.

-nl <eof> zakańcza *login* (domyślna)

-p wyświetla *prompt* (prompt domyślny jest $)

-p=<ciąg> ustanawia *prompt* bieżącego *shell* na ciąg =<ciąg>.

-np wstrzymuje wyświetlanie *prompt*.

-t powoduje echo potoków wejściowych.

-nt nie powoduje echa potoków wejściowych. (domyślna)

-v powoduje wyświetlanie komunikatów dla każdego katalogu podczas wykonywania polecenia.

-nv wyłącza komunikaty (domyślna)

-x przerywa proces po wystąpieniu błędu (domyślna)

-nx nie przerywa procesu po wystąpieniu błędu.

Każda z powyższych opcji może być podana bez dodatkowo formułowanego polecenia. Może ona też być wprowadzona przez użycie specjalnego polecenia <set> i z pominięciem wtedy znaku (-), (myślnik).

2.5.1 Środowisko shell

Shell utrzymuje jednoznaczną listę zmiennych środowiska dla każdego użytkownika. Każda zmiana środowiska na głębszym poziomie *shell* (następca) nie zmienia środowiska przodka. Wynika stąd charakter globalny tych zmiennych. Cztery zmienne są ustawiane automatycznie po zalogowaniu użytkownika do systemu (w oparciu o plik *.login* danego użytkownika). Sa to: PORT specyfikująca nazwę terminala, HOME specyfikująca katalog *home*, tzn. katalog pierwotny po zalogowaniu się użytkownika oraz ten, do którego powraca się podając polecenie *chd* bez parametrów, SHELL czyli pierwszy proces po uruchomieniu systemu oraz USER czyli nazwa użytkownika podawana

podczas logowania. Inne ważne zmienne w systemie to: PATH podająca zestaw dostępnych katalogów (oddzielonych dwukropkiem), PROMPT określająca sposób ponaglania, _sh specyfikująca numer poziomu bazowego dla *shell*, oraz TERM specyfikująca typ stosowanego terminala.

Każda z powyższych zmiennych moze bć zmodyfikowana poleceniem <setenv>, a pełne środowisko zostanie wylistowane po podaniu polecenia <printenv>. Usunięcie zmiennej z pamięci uzyskuje się poleceniem <unsetenv>.

2.5.2 Wprowadzanie poleceń w shell

Polecenie <....> dla shell zawiera:

1. słowo kluczowe (nazwa programu, plik proceduralny itp),
2. parametry (nazwy plików, programy, wartości, zmienne, stałe parametry itp),
3. modyfikatory wykonawcze (modyfikowanie wykonania programu przez skierowanie (redirection) WE/WY oraz zmianę priorytetu). Modyfikatory to:

#	dodatkowy obszar pamięci,
^	priorytet procesu,
>	skierowanie wyjścia,
<	skierowanie wejścia,
>>	skierowanie wyjścia błędu.

4. separatory (specyfikują, czy określone programy mają być wykonywane sekwencyjnie, czy współbieżnie). Separatory to:

;	przetwarzanie sekwencyjne,
&	przetwarzanie współbieżne,
!	przetwarzanie potokowe (pipeling)

Uwaga' & na końcu polecenia oznacza żądanie jego wykonania w tle.

Niezależnie od przetwarzania określonego ww. separatorami, lub przetwarzania indywidualnego, stosuje się przetwarzanie grupowe, poprzez zastosowanie nawiasów w poleceniu np:

```
(dir /d0; dir/d1) >/p
```

2.5.3 Procedury .login oraz .logout

Zalogowanie się użytkownika do systemu (polecenie <login>) powoduje, po wykonaniu programu inicjującego określonego przez 7. pole linii właściciela konta w pliku **/dd/sys/password**, automatyczne wykonanie poleceń zebranych w pliku **../.login**. Umożliwia to samoczynne wykonanie wszelkich poleceń inicjujących, określonych do wykonania przez użytkownika (patrz pktt. 1.5.1). Podobnie, po poleceniu <logout> wykonuje się zestaw instrukcji zawarty w pliku **../.logout**. Procedury **.login** i **.logout** są umieszczone w katalogu HOME użytkownika. Procedury te są różne dla każdego z użytkowników.

2.6 Funkcje użytkowe dołączone do systemu

Funkcje użytkowe dołączone do systemu uzupełniają jądro systemu. Są to:

1. Funkcje podstawowe
2. Funkcje programowania, oraz
3. Funkcje zarządzania systemem.

Do funkcji podstawowych zalicza się:

attr	backup	build	chd	chx	copy	date
del	deldir	dir	dsave	echo	edt	format
free	help	kill	list	makdir	merge	mfree
pd	pr	procs	rename	set	setime	shell
w	wait					

Do funkcji programowania zalicza się:

binex	cfp	cmp	code	compress	count	dump
ex	exbin	expand	freestore	fsave	grep	load
logout	make	printenv	qsort	save	setenv	tape
tee	touch	tmode	tr			

Do funkcji zarządzania systemem zalicza się:

break	dcheck	deiniz	devs	events	fixmod	ident
iniz	irqs	link	login	mdir	moded	os9gen

romsplit setpr

Dokładny opis funkcji zawarty jest w dokumentacji OS-9 Utilities i będzie przedmiotem szczegółowej analizy w etapie 4 zlecenia S1300. Obecnie zostanie przedstawiony jedynie zakres funkcjonalny tych funkcji (* - funkcje wbudowane w shell):

attr	Sprawdzanie lub zmiana atrybutów dostępu do pliku,
backup	Kopiowanie sektor po sektorze, niezależnie od struktury,
binex	Zamiana postaci binarnej pliku do postaci S-record,
break	Zatrzymanie i wycofanie system OS-9,
build	Kreowanie pliku o danej ścieżce,
cfp	Tworzenie przejściowego pliku proceduralnego,
chd	*Zamiana katalogu roboczego,
cmp	Porównanie binarne dwóch plików,
code	Drukowanie znaku z jego wartością hexadecymalną,
compress	Kompresja pliku tekstowego,
copy	Kopiowanie plików,
count	Zliczanie znaków w pliku,
date	Wskazanie bieżącej daty,
dcheck	Testowanie dysku,
deiniz	Deinicjalizowanie dołączonego wcześniej urządzenia,
del	Usuwanie pliku,
deldir	Usuwanie katalogów z ich zawartością,
devs	Drukowanie listy aktywnych urządzeń systemu,
dir	Drukowanie listy nazw plików,
dsave	Kopiowanie bitowe (backup) lub strukturalne,
dump	Wyświetlanie fizycznej zawartości pliku,
echo	Powtarzanie argumentów funkcji na standardowym wyjściu,
edt	Edytor liniowy,
events	Wyświetlanie listy aktywnych zdarzeń,
ex	*Startowanie wykonywania kolejnego programu,
exbin	Zamiana postaci S-record pliku do postaci binarnej,
expand	Odtwarzanie pliku poddanego uprzednio kompresji,
fixmod	Weryfikowanie i modyfikowanie parzystości i CRC modułu,
format	Fizyczne inicjalizowanie struktury dysku,
free	Wyświetlanie zajętości dysku,

frestore	Odtwarzanie struktury katalogów z taśmy lub z dysku,
fsave	Zapis struktury katalogów na taśmie lub dysku,
grep	Poszukiwanie ciągu znaków ww wskazanej ścieżce,
help	Wyświetlanie informacji dotyczącej określonej funkcji,
ident	Wyświetlanie informacji nagłówkowych modułu,
iniz	Inicjalizowanie urządzenia systemu,
irqs	Wyświetlanie tablicy wprowadzonych IRQ,
kill	*Bezwarunkowe usuwanie procesu z listy zadań systemu,
link	Konsolidowanie załadowanych do pamięci modułów,
list	Wyświetlanie zawartości pliku,
load	Ladowanie wyspecyfikowanych modułów do pamięci,
login	Wprowadzanie początkowych wartości parametrów systemu,
logout	Przeprowadzenie operacji przy opuszczaniu systemu,
makdir	Tworzenie katalogu,
make	Nadzorowanie modyfikacji plików oraz kompilacji,
mdir	Wyświetlanie nazw aktualnych modułów w systemie,
merge	Zestawianie kilku plików w jeden,
mfree	Wyświetlanie obszarów pamięci aktualnie nie zajętej,
moded	Edytowanie wybranych pól modułów systemu,
os9gen	Tworzenie i linkowanie pliku startowego OS9Boot,
pd	Wyświetlanie ścieżki do katalogu roboczego,
pr	Formatowanie listingu plików dla STDOUT,
printenv	Wyświetlanie zmiennych środowiska systemu,
procs	Wyświetlanie listy bieżących procesów,
qsort	Algorytm sortowania,
rename	Przypisywanie plikowi nowej nazwy,
romsplit	Rozkładanie pliku na dwa, lub cztery podpliki,
save	Kopiowanie modułu z pamięci do bieżącego katalogu,
set	*Ustawianie wartości środowiska shell,
setenv	Ustawianie wartości shell dla programu-następcy,
setime	Ustawianie daty i czasu systemu,
setpr	Zmiana priorytetu CPU procesu,
shell	Interpreter poleceń systemu OS-9,
sleep	Zawieszanie procesu na określoną liczbę cykli,
tape	Zapis i odczyt danych na taśmie,
tee	Przekazwanie komunikatów do użytkowników systemu,

tmode	Zmiana parametrów terminala użytkownika,
touch	Modyfikowanie daty ostatnich zmian pliku,
tr	Zamiana określonych ciągów znakowych w pliku,
tsmon	Zarządzanie nieaktywnymi terminalami,
unlink	Zwalnianie zbędnych modułów pamięci,
unsetenv	Usuwanie zmiennych z listy środowiska systemu,
xmode	Inicjalizacja parametrów monitora, drukarki i RS232,

2.7 Edytor uMACS (mikromacs)

2.7.1 Podstawowe cechy edytora

uMACS jest ekranowym edytorem tekstu stosowanym do tworzenia i modyfikowania tekstu. Umożliwia pracę z wieloma plikami jednocześnie w tym samym czasie, z natychmiastowym wyborem dostępu do określonego pliku.

Polecenia uMACS mogą być wykonywane na dwa sposoby:

1. Poprzez wprowadzenie, po wciśnięciu klawisza <Esc>, nazwy polecenia, oraz
2. Poprzez podanie zdefiniowanej sekwencji klawiszów, odpowiadającej odpowiedniemu poleceniu.

I tak np. dla polecenia przesunięcia kursora do początku pliku, wymienione sposoby wymagają zapisu:

ad 1. <Esc> beginning-of-file, lub

ad 2. ^ <.

Edytor uMACS zawiera 91 poleceń, ale nie wszystkie z nich mają odpowiedniki sekwencji klawiszów. W takich przypadkach ich wywołanie następuje poprzez podanie pełnej nazwy polecenia.

2.7.1.1 Polecenia MACRO

Wprowadzanie poleceń macro można realizować poprzez:

1. Dynamiczne wprowadzanie zestawu poleceń rozpoczynającego się poleceniem <begin-macro> i kończące się poleceniem <end-macro>,
2. Wydanie polecenia <execute-file> [nazwa.pliku], które realizuje kolejno polecenia zawarte w pliku o nazwie [nazwa.pliku],

3. Wykonywanie statyczne macro poprzez utworzenie pliku tekstowego o nazwie .umacsrc w katalogu HOME, w którym należy umieścić zestaw wymaganych poleceń. Wywołanie edytora poleceniem <umacs> powoduje sprawdzenie obecności pliku .umacsrc, i jeśli on istnieje, jego wykonanie.

2.7.1.2 Dostęp do programu shell oraz plików systemu

Dostęp do programu shell może nastąpić bez przerywania pracy edytora, za pomocą poleceń <i-shell> (^XC), lub <shell-command> (^X!). Dostęp do plików spod edytora umożliwia operowanie ich zasobami tak, jak spod programu shell.

2.7.1.3 Mody pracy edytora

Edytor może pracować w niżej wymienionych modach pracy:

1. OVER - przepisywanie tekstu w miejsce już istniejącego,
2. EXACT - wyszukiwanie tekstu z uwzględnieniem wielkości liter,
3. WRAP - przenoszenie tekstu do kolejnej linii przy przekroczeniu marginesu,
4. CMODE - mod dostosowany do edycji programów pisanych w języku C (z rozszerzeniem .c lub .h). Powoduje wprowadzanie wcięć podczas edycji programów zgodnie ze zwyczajami przyjętymi dla tego języka. Odpowiednio reaguje na znaki {, } oraz #.
5. VIEW - oglądanie pliku bez możliwości jego korekty.

2.7.1.5 Ogólny podział poleceń edytora umacs.

Polecenia, szczegółowo opisane w instrukcji Using uMACS, dzielą się na polecenia umożliwiające:

1. pozycjonowanie kursora,
2. wprowadzanie tekstu,
3. usuwanie tekstu,
4. wyszukiwaie i zamianę tekstu,
5. operacje na blokach,
6. formatowanie,

7. operacje na buforach, oraz
8. operacje na oknach.

Polecenia te wykonywane są w każdym z wybranych modów pracy, lub przy ich odpowiedniej kombinacji.

3. Zestaw narzędzi programowania minikomputera w języku C

3.1. Wprowadzenie.

Kompilator C firmy Microware jest wysokiej jakości programem narzędziowym o następujących cechach:

- pełna implementacja języka C,
- wydajne generowanie szybkiego i zwartego kodu,
- generowanie kodu współużywalnego, niezależnego od adresu ładowania, z przeznaczeniem dla pamięci typu ROM,
- wysoka szybkość kompilacji,
- kompatybilność bibliotek standardowych systemów OS-9 i UNIX.

3.2. Instalacja i uruchamianie kompilatora.

Następujące zbiory binarne, tworzące wykonywalną część systemu kompilacji, należy skopiować z dyskietek dystrybucyjnych do katalogu CMDS standardowego dysku systemowego:

cc	program zarządzjący przebiegiem kompilacji
cpp	makro-preprocesor
c68	kompilator
o68	optymalizator
c68	kompilator dla procesora 68020

Następujące dodatkowe moduły wymagane są również do przebiegu kompilacji:

r68	makroassembler OS-9/68000
l68	linker OS-9/68000
cio	pułapkowy handler biblioteki I/O
r68020	makroassembler OS-9/68020
cio020	pułapkowy handler biblioteki I/O dla CPU 68020

3.2.1 Biblioteki.

Następujące zbiory należy skopiować z dyskietek dystrybucyjnych do katalogu LIB standardowego dysku systemowego:

cstart.r	Kod inicjujący dla kompilowanego programu.
clib.r	Biblioteka standardowa, matematyczna i systemowa.
clibn.r	Identyczny z clib.r poza tym, że nie obejmuje zmiennoprzecinkowych funkcji matematycznych w trybie obsługi pułapkowej (mogą być dołączane z biblioteki math.l).
clib020	Jak clib.l, lecz dla CPU 68020.
clib020n.l	Jak clibn.l, lecz dla CPU 68020.
clib020h.l	Jak clib020n.l, lecz zawiera instrukcje odwołujące się do koprocesora 68881 dla operacji zmiennoprzecinkowych.

Następujące zbiory, również wymagane do przebiegu kompilacji, mogą być umieszczone w dowolnym katalogu:

math.l	Zawiera te same matematyczne funkcje zmiennoprzecinkowe, co funkcje obsługiwane w trybie pułapkowym, lecz w formie wolnostojącej (wprost dołączane).
math881.l	Jak math.l, lecz korzysta z instrukcji koprocesora 68881 przy operacjach zmiennoprzecinkowych.
sys.l	Definicje systemowe.
termlib.l	Funkcje obsługi terminala ekranowego.

3.2.2 Definicje.

Przedstawione poniżej zbiory, wykorzystywane przez funkcje biblioteczne, definiują stałe, struktury danych, makrodefinicje. Zbiory te winny być skopiowane z dyskietek dystrybucyjnych do katalogu DEFS standardowego dysku systemowego:

ctype.h	dir.h	direct.h	errno.h	events.h
math.h	procid.h	setjmp.h	setsys.h	systat.h
signal.h	stdio.h	strings.h	termcap.h	time.h
types.h				

3.2.3 Użytkowanie zasobów środowiska kompilatora języka C.

3.2.3.1 Wyszukiwanie zbiorow bibliotecznych i definicji.

Przebieg kompilacji programu opiera się na dostępie do zbiorów definicji i bibliotek. Przechowywane są one odpwiednio w katalogach DEFS i LIB dysku systemowego, w zależności od konfiguracji sprzętowej następująco:

system z dyskiem stałym	-	/h0/DEFS i /h0/LIB
system bez dysku stałego	-	/d0/DEFS i /d0/LIB
system z RAM-dyskiem	-	/r0/DEFS i /r0/LIB lub
		/dd/DEFS i /dd/LIB

Położenie bibliotek określane jest jedną z trzech metod, przy zachowaniu następującego priorytetu:

opcja w linii wywołania polecenia,

zmienne środowiska powłoki systemu,

standardowa metoda wyszukiwania.

Opcja -r umieszczona w lini wywołania kompilatora specyfikuje wyszukiwanie zbiorów definicji w dodatkowym katalogu w stosunku do DEF (przeszukiwanie dokonywane jest przed przeszukaniem katalogu DEF). Opcja -w powoduje zastąpienie standardowego katalogu LIB, katalogiem podanym w tej opcji.

Zmienne środowiska powłoki CLIB i CDEF również mogą być używane do określenia katalogów przechowujących odpowiednio biblioteki i zbiory definicji (zmienne te ustawiane są programem setenv).

Standardowe katalogi DEFS i LIB powinny znajdować się na pierwszym z prezentowanej niżej listy przeszukiwania urządzeń dyskowych, znajdującym się w systemie:

/dd	urządzenie standardowe (zwykle RAM-dysk)
/h0	dysk twardy
/d0	dysk miękki.

3.2.3.2 Wywołanie kompilatora.

Przebiegiem kompilacji zarządza program **cc**. Przyjmuje on opcje kompilacji umieszczone w lini wywołania programu i steruje kolejnymi fragmentami procesu kompilacji.

Postać wywołania kompilatora:

cc [<opcje>] <nazwa_zbioru> [<nazwa_zbioru>] [<opcje>]

Pojedyńczym wywołaniem kompilatora dokonuje się kompilacji jednego lub więcej zbiorów, przy czym w jednej lini wywołania można umieszczać nazwy zbiorów źródłowych pisanych w języku C, w assemblerze i modułów relokowalnych.

Kompilacja obejmuje cztery przebiegi:

wstępne przetwarzanie kodu,

tworzenie kodu assemblerowego,

tworzenie modułów relokowalnych,

tworzenie wykonywalnego modułu ładowania.

Kompilator akceptuje trzy typy zbiorów źródłowych, zachowując następującą konwencję:

rozszerzenie nazwy	opis
.c	zbiór źródłowy w C
.a	zbiór źródłowy assemblerowy
.r	moduł relokowalny

Konwencja uzyskiwania wyjściowego zbioru kompilacji jest następująca: kompilując pojedyńczy zbiór, moduł ładowania otrzymuje nazwę zbioru kompilowanego, po odcięciu rozszerzenia jego nazwy; umieszczając w poleceniu kompilacji wiele zbiorów, zbiór wyjściowy otrzymuje nazwę **output**, o ile nie użyto alternatywnej opcji. W obu przypadkach zbiór wyjściowy zakładany jest standardowo w katalogu bieżącym. Wszystkie moduły relokowalne, generowane jako produkty pośrednie kompilacji, umieszczane są w tych samych katalogach, co odpwiadające im zbiory źródłowe, z nazwami o rozszerzeniu .r.

3.2.3.3 Zbiory tymczasowe i optymalizacja kompilacji.

Podczas kompilacji zbiory tymczasowe tworzone są w katalogu bieżącym. Jako zgrubną regułę można przyjć, że należy zarezerwować

na nie co najmniej 3 x tyle miejsca, ile zajmuje najdłuższy zbiór źródłowy wraz z jego zbiorami definicji (.h). Prędkość kompilacji zależy przede wszystkim od szybkości zapisu/odczytu zbiorów tymczasowych. Zatem katalog bieżący winien znajdować się na najszybszym urządzeniu dyskowym.

3.2.3.4 Opcje selekcji bibliotek C.

Zastosowanie opcji -i w kompilacji powoduje, że zainstalowany systemowy moduł handlera pułapkowego cio używany będzie do wykonywania funkcji I/O, zamiast włączania kodów tych funkcji do linkowanego programu. Moduł cio jest automatycznie ładowany podczas startu systemu.

3.2.3.5 Opcje kompilacji.

Opcje rozpoznawane są przed przebiegiem kompilacji, mogą więc być umieszczane w dowolnym miejscu lini wywołania kompilatora. Wielkość liter opcji nie jest istotna. Ponadto mogą być one grupowane (np. -sr), o ile opcja nie obejmuje argumentów.

opcja	opis
-a	Kończy działanie przed fazą assemblacji, generując produkt końcowy w postaci zbioru assemblerowego o nazwie z rozszerzeniem .a.
-bg	Ustawia bit "sticky" w nagłówku modułu ładowania powodujący, że moduł zostaje w pamięci nawet po wyzerowaniu licznika podwiązania.
-bp	Drukowane są argumenty przekazywane do poszczególnych faz kompilacji i status wyjścia.
-c	Generuje kod źródłowy jako komentarze do kodu assemblerowego; użyteczne z opcją -a.
-d<ident>	Równoważne użyciu **#define <ident>** w zbiorze źródłowym.
-e=<numb>	Ustawia bajt numeru edycji.
-f=<pathlist>	Zamazuje konwencję nazwy zbioru wyjściowego

kompilacji poprzez narzucenie nazwy ostatniego elementu <pathlist>. Nie można używać łącznie z opcjami -a, -r. Jeśli <pathlist> jest ścieżką względną, odnosi się do bieżącego katalogu wykonawczego.

-fd=<pathlist> Zgodne z opcją -f=<pathlist>; wyjątek - gdy <pathlist> jest ścieżką względną, odnosi się do bieżącego katalogu danych.

-g Linker generuje moduł symboli dla symbolicznego assemblerowego debagera (zbiór z rozszerzeniem nazwy .stb), umieszczany w katalogu STB, jeśli taki istnieje, lub w tym samym katalogu co zbiór wyjściowy kompilacji.

-i Włącza do programu bibliotekę cio.l powodując, że wywołania wybranych funkcji C I/O wykonywane będą przez moduł handlera pułapkowego cio.

-j Zabezpiecza przed tworzeniem tablicy skoków przez linker.

-k=<n>[w|l][cw|cl][f]

<n> docelowy procesor: 0 = 6800(standard)
2 = 68020

w 16-bitowy offset danych (standard 68000)

l 32-bitowy offset danych (standard 68020)

cw 16-bitowa adresacja kodu (standard 68000)

cl 32-bitowa adresacja kodu (standard 68020)

f generowanie instrukcji 68881 (dla 68020)

-l = <path> Określa zbiór biblioteczny do przeszukania przed biblioteką standardową, matematyczną i systemową.

-m=<mem_size> Przydziela <mem_size> pamięci na stos.

-n=<name> Określa nazwę modułu końcowego.

-o Nie optymalizuje kodu. Zalecane przy debagowaniu programu.

-q Brak informacji o wykonywanych krokach pośrednich kompilacji; komunikaty błędów wyprowadzane.

-r[=<dir>] Kompilator kończy pracę przed fazą linkowania.

Jeśli -r=<dir>, moduły relokowalne umieszczane są w katalogu <dir>.

-s Wstrzymanie generowania kodu kontroli przepełnienia stosu.

-t=<dir> Zbiory tymczasowe umieszczane są w katalogu <dir>. Jeśli RAM-dysk zawiera ten katalog (tj. -t=/r0), czas kompilacji będzie zredukowany.

-u<name> Likwiduje uprzednio zdefiniowane makrodefinicje OSK i mc68000, pozwalające zidentyfikować kompilator, którym kompiluje się program.

-v=<dir> Określa dodatkowy katalog do przeszukiwania zbiorów definicji (.h). Zakłada się, że zbiory o nazwach umieszczonych pomiędzy znakami (") znajdują się w katalogu bieżącym, zaś umieszczone w nawiasach ukośnych (<>) - w katalogu <dir>.
Opcja może pojawić się więcej niż jeden raz. Przeszukiwanie katalogów odbywa się w kolejności ich pojawienia się w lini wywołania kompilatora. Standardowy katalog DEF przeszukiwany jest jako ostatni.

-w=<dir> Określa katalog zawierający standardowe zbiory biblioteczne (cstart.r, clib.l, ...).

-x Generowanie instrukcji pułapkowych, realizujących dostęp do zmiennoprzecinkowych funkcji matematycznych. Opcja powinna być użyta zarówno przy kompilacji, jak i linkowaniu (jeśli fazy te realizowane są oddzielnie).

3.2.4 Przykłady wywołań kompilatora.

postać wywołania	produkt wyjściowy (katalog)
działanie kompilatora	
cc prg.c	prg (katalog bieżący wykonawczy)
Wygenerowanie modułu wykonywalnego	

cc prg.c -a prg.a (katalog bieżący danych)

Wygenerowanie assemblerowego zbioru źródłowego

cc prg.c -r prg.r (bieżący katalog danych)

Wygenerowanie modułu relokowalnego

cc prg1.c prg2.c prg1.r, prg2.r (bieżący katalog danych);
outout (katalog wykonawczy)

Wygenerowanie modułu wykonywalnego

cc prg1.c /d0/ted/prg2.c prg1.r (bieżący katalog danych)
prg2.r (/d0/ted)
outut (katalog wykonawczy)

Wygenerowanie modułu wykonywalnego

cc -e=3 name.c -f=prog name.r (bieżący katalog danych)
prog (katalog wykonawczy)

Wygenerowanie modułu wykonywalnego z poziomem przeglądania ustawionym na 3.

cc sieve.c -igf=sieve sieve (katalog wykonawczy)
sieve.stb (bieżący katalog danych)

Wygenerowanie modułu wykonywalnego z wykorzystaniem modułu funkcji cio, a także modułu symboli dla debagera.

cc prg.c -dfloats prg (katalog wykonawczy)

Wygenerowanie modułu wykonywalnego z przekazaniem do kompilatora zdefiniowanego identyfikatora floats.

cc prg.c -k2 prg (katalog wykonawczy)

Kod dla procesora 68020.

cc prg.c -k2w prg (katalog wykonawczy)

Kod dla procesora 68020 z użyciem adresacji z 16-bitowym offsetem.

cc prg.c -k2f prg (katalog wykonawczy)

Kod dla procesorów 68020/68881.

3.2.5 Biblioteki matematyczne i selekcje opcji.

Opcje -k i -x określają, które z bubliotek używane są podczas kompilacji:

wywołanie	procesor	bibl.C	bibl. mat.	opis
cc	68000	clib.n	math.l	Wybór standardowy. Kod zmiennoprzecinkowych funkcji matematycznych włączany do do programu.
cc -x	68000	clib.l	<trap>	Wykorzyystywany jest matematyczny handler pułapkowy.
cc k2	68020	clib020n.l	math.l	Jak wywołanie cc poza tym, że inny procesor.
cc -k2 -x	68020	clib020.l	<trap>	Jak wywołanie cc -x poza tym, że inny procesor.
cc -k2f	68020/881	clib020h.l	math881.l	Generowane są instrukcje zmiennoprzecinkowe dla koprocesora 68881.

3.3. Implementacja kompilatora.

3.3.1 Reprezentacja danych.

typ danych	bajty	reprezentacja wewnętrzna
char	1	BU2
unsigned char	1	B
short	2	BU2
unsigned short	2	B
int	4	BU2
unsigned	4	B
long	4	BU2
float	4	BF
double	8	BF
"pointer do.."	4	adres

BU2 - binarne uzupełnienie do 2

B - binarana bez znaku

BF - binarna zmiennoprzecinkowa

Kompilator zachowuje konwencję zamiany typów **char** i **short** na **int** z powieleniem znaku.

3.3.1.1 Zmienne rejestrowe.

Rozmiar kodu programu i szybkość jego wykonywania mogą byyć zoptymalizowane przy pomocy zmiennych typu **register**. Szczególnie efektywność ich użycia uzyskuje się w odniesieniu do pointerów i liczników pętli.

Deklaracje rejestrowe można stosować do zmiennych automatycznych i argumentów funkcji. Niedopuszczalne deklaracje są ignorowane. W ramach każdej funkcji dostępne są trzy 32-bitowe rejestry adresowe na pointery i trzy 32-bitowe rejestry danych na zmienne nie będące pointerami. Przyporządkowywanie rejestrów dokonywane jest w kolejności deklaracji zmiennych.

3.3.1.2 Format zmiennoprzecinkowy.

Format pojedyńczej precyzji (float):

(MSB) (LSB)

znak(1) wykładnik potęgi(8) mantysa(23)

precyzja: 7 cyfr (w nawiasach długość pola w bitach)

Format podwójnej precyzji (double)

(MSB) (LSB)

znak(1) wykładnik potęgi(11) mantysa(52)

precyzja: 11 cyfr

Mantysa ma prowadząca stałą jedynkę. Zmienne typu **float** są zamieniane do postaci **double** przy przekazywaniu parametrów do funkcji. Operacje zmiennoprzecinkowe realizowane są przez zmiennoprzecinkowy handler pułapkowy.

3.3.1.3 Dostęp do parametrów lini wywołania programu (Argc, Argv, Envp).

Standardowe argumenty języka C **argc** i **argv** dostępne są w **main** zgodnie z opisem w książce **K&R**. Argument **envp** wskazuje na listę zmiennych środowiska procesu, normalnie ustawianą przez **shell** systemu.

3.3.1.4 Wstawki assemblerowe.

Kompilator umożliwia wprowadzanie do tekstu źródłowego C sekwencji instrukcji assemblerowych. Linia rozpoczynająca się **#asm** przestawia kompilator w tryb, w którym przekazuje on kolejno napotkane linie wprost do assemblerowego produktu wyjściowego, aż do napotkania linii zaczynającej się od **#endasm**, przełączającej z powrotem kompilator do trybu normalnego. Identyczny efekt uzyskuje się, rozpoczynając pojedyńcze linie pisane w assemblerze znakiem @

3.3.1.5 Sekwencje ze znakiem sterującym Esc.

Sekwencje ze znakiem **Esc** dla stałych znakowych i ciągów znakowych (stringów) są rozszerzone w stosunku do **K&R** następująco:

linefeed (LF):	\l	
wzorzec bitów:	\NNN	(stała oktalna)
	\dNNN	(stała decymalna)
	\xNNN	(stała hexadecymalna).

3.3.1.6 Klasa pamięci remote.

Tryb adresowania pośredniego poprzez rejestr z przesunięciem ograniczony jest dla procesora 68000 do 64K. Jeśli wymaga się dostępu do obszaru pamięci większego niż 64K, musi być użyty tryb indeksowanego adresowania pośredniego poprzez rejestr z przesunięciem, co umozliwia indeksowanie rejestru adresowego 32-bitowym offsetem. Kompilator generuje odwołania do pamięci przy użyciu 32-bitowego offsetu dla zmiennych deklarowanych jako **remote**. Wszystkie dane typu **remote** przydzielane są przez linker po zakończeniu alokacji zwykłych danych. Należy pamiętać, że użycie danych typu **remote** spowalnia wykonywanie programu i wydłuża jego rozmiar.

Alternatywną metodą w stosunku doposługiwania się danymi oddalonymi jest deklarowanie w programie pointerów do dużych tablic i wykonanie odwołania do systemu, przydzielającego potrzebny obszar pamięci.

Opcja -k kompilatora powoduje, że kompilator będzie generować przy dostępie do danych wszystkie offsety jako 16-bitowe lub wszystkie jako 32-bitowe.

Opis deklaracji typu **remote:**

remote <typ_danych> <zmienna>

Przykład:

remote double bigarray[10000];

Powyższa deklaracja rezerwuje 80000 bajtów pamięci dla 10000 elementów tablicy.

3.4 Organizacja kompilatora.

Kompilator C składa się z trzech głównych części:

cpp	preprocesor
c68	kompilator
o68	optymalizator

Makroassembler (r68) i linker (l68) są niezbędne do assemblacji i połączenia produktu wyjściowego kompilatora. Program zarządzający (cc) stanowi interfejs pomiędzy wywołaniem użytkownika, a poszczególnymi fazami kompilacji.

Preprocesor przygotowuje zbiór źródłowy do kompilacji. W następnej fazie kompilator tłumaczy linie źródłowe, będące produktem końcowym preprocesora, na źródłowy kod assemblerowy do przetworzenia przez makroassembler. Opisy ewentualnych błędów kompilacji przekazywane są do standardowego strumienia błędów (STDER).

3.4.1 Moduł ładowalny.

Kompilator generuje kod współużywalny, niezależny od adresu ładowania, przeznaczony dla pamięci typy ROM.

3.4.2 Stos.

Na stos rezerwowana jest część obszaru danych programu o najwyższych adresach, standardowo około 2K. W przypadku potrzeby, powiększenie rozmiaru stosu dokonuje się opcją -m kompilatora.

Kompilator generuje kod sprawdzający przepełnienie stosu przy każdym wywołaniu funkcji. Opcja -s kompilacji stosowana jest do wyłączenia powyższej funkcji kompilatora.

3.5 Wykorzystanie programu make

3.5.1 Istota programu

Wiele rodzajów zbiorów, w fazie ich generowania, jest uzależnionych od innych zbiorów. Jeśli zbiory tworzące produkt końcowy ulegną zmianie, wynikowy zbiór staje się nieaktualny. Program make przeznaczony jest do automatycznej obsługi, podczas generowania zbioru uzależnionego od innych, jedynie tych zbiorów, które uległy modyfikacji. Posiada wbudowane mechanizmy domyślne, przeznaczone do kompilacji programów w języku wyższym. Make obsługuje zbiory przy użyciu specjalnego typu zbioru, o nazwie **makefile**, opisującego związek pomiędzy zbiorem końcowym, a zbiorami tworzącymi go. Produkt końcowy zwany będzie dalej zbiorem docelowym, zaś tworzące go zbiory - zbiorami zależnościowymi. Makefile zawiera trzy rodzaje opisów:

opisy zbiorów zależnościowych,

polecenia systemowe,

komentarze.

Opis zbiorów zależnościowych ustala związek zbioru docelowego ze zbiorami zależnościowymi, użytymi do budowy zbioru docelowego. Postać opisu jest następująca:

<zbiór_docelowy>:[[<zbiór_zależnościowy>],<zbiór_zależnościowy>]

Lista zbiorów następująca po zbiorze docelowym zwana jest listą zależności. **Makefile** może zawierać dowolną liczbę opisów zależności. Zbiór zależnościowy z jednego opisu może być zbiorem docelowym w drugim opisie zależności. Każdy **makefile** zawiera tylko jeden główny zbiór docelowy, umieszczany zwykle w pierwszym opisie zależności.

W liniach poleceń systemowych wprowadza się polecenia, które muszą być wykonane celem uaktualnienia zbioru docelowego. Wykonywane są one, jeśli zbiory zależnościowe są młodsze, niż zbiór docelowy. W przypadku braku poleceń, **make** będzie próbował przeprowadzić standardowe, domyślne operacje. **Make** rozpoznaje polecenia w liniach zaczynających się od co najmniej jednej spacji lub znaku tabulacji. Akceptowane są dowolne polecenia systemu OS-9. Dla każdego opisu zależności można wprowadzić dowolną liczbę poleceń. Polecenia można przenosić do następnej lini (zachowując konwencję prowadzących spacji) znakiem backslash (\).

Przykład:

```
<zbiór_docelowy>:[[<zbiór>],<zbiór>]
        <polecenie OS-9>
        <polecenie OS-9> \
        <kontynuacja linii poleceń>
```

Komentarz tworzą linie rozpoczynające się znakiem asterisk (*) lub w środku lini sekwencje znaków poprzedzone znakiem hash (#), jeśli nie występuje bezpośrednio po nim cyfra. Linie puste są ignorowane.

Przykład:

```
<zbiór_docelowy>:[[<zbiór>],<zbiór>]
        * komentarz
        * komentarz
        <polecenie OS-9>        # komentarz
```

Opisy zależności i polecenia mogą być kontynuowane w następnej lini, po umieszczeniu znaku backslash (\) poprzedzonego spacją. Pojedyńcza linia zawiera do 256 znaków.

Przykład:

```
FILE: aaa.r bbb.r ccc.r \
ddd.r fff.r
      touch aaa.r bbb.r ccc.r \
      ddd.r fff.r
```

3.5.2 Wywołanie programu make.

Program **make** wykonuje polecenia umieszczone w zbiorze **makefile**. Wywołany poleceniem nie zawierającym w lini wywołania nazwy zbioru docelowego, za zbiór docelowy uzna pierwszy zbiór docelowy umieszczony w zbiorze **makefile**. Wywołany poleceniem zawierającym jeden lub więcej zbiorów docelowych, odczyta i przetworzy **makefile** oraz spróbuje wygenerować jedynie powyższe zbiory. **Make** przetwarza zbiór **makefile** trzykrotnie. W pierwszym przebiegu zestawia tablicę zależności, umieszczając w niej zbiór docelowy i zbiory zależnościowe wprost, tj. w/g list zależności. Następnie tworzy łańcuch powiązań, znajdując zbiory docelowe umieszczone na listach zależności. W drugim przebiegu **make** próbuje rozwiązać istniejące zależności domyślne (opisane poniżej). W trzecim przebiegu program przeglądając listę zależności, porównuje daty założenia zbiorów. Po znalezieniu zbioru młodszego niż zbiór docelowy, powoduje wykonanie wyspecyfikowanych poleceń systemowych. Jeśli takowych w zbiorze **makefile** nie umieszczono, **make** generuje polecenia systemowe w/g reguł przedstawionych w dalszej części rozdziału. Komunikaty o wykonywanych poleceniach systemowych przekazywane są do standardowy strumienia wyjściowego. **Make** zatrzymywany jest po wykonaniu polecenia zakończonego błędem.

Opis wywołania programu:

make [<opcje>] [<zb_docel>] {[<zb_docel>]} [<makrodefinicje>]

3.5.3 Definicje domyślne.

Przy wykonywaniu programu **make**, zbiór docelowy uznawany jest domyślnie za program do kompilacji. Jeśli nie odpowiada to rzeczywistości, niezbędne do wykonania polecenia systemowe muszą być wprost umieszczone dla każdej listy zależności w zbiorze **makefile**.

Make używa następujących definicji i reguł do samodzielnego generowania poleceń systemowych:

zbiory ładowalne	Zbiory o nazwach bez rozszerzeń. Tworzone ze zbiorów relokowalnych - linkowane, kiedy zachodzi taka konieczność.
zbiory relokowalne	Zbiory o nazwach z rozszerzeniem .r. Tworzone ze zbiorów źródłowych - kompilowane lub assemblowane, jeśli zachodzi taka konieczność.
zbiory źródłowe	Zbiory o nazwach z rozszerzenie: .a, .c, .f lub .p.
standardowy kompilator	cc
standardowy assembler	r68
standardowy linker	l68
standardowy katalog	dla wszystkich zbiorów - bieżący katalog danych

3.5.4 Rozpoznawanie makrodefinicji.

Poza rozpoznawaniem reguł kompilacji, **make** rozpoznaje pewne makrodefinicje. Muszą być one poprzedzone znakiem dolara ($), jeśli zaś ich nazwa jest dłuższa niż pojedynczy znak, dodatkowo powinny być ujęte w nawias. Na przykład, $R odpowiada makrodefinicji R, $(PFLAGS) odpowiada makrodefinicji PFLAGS, zaś $(B) i $B - makrodefinicji B.

Makrodefinicje mogą być umieszczane w zbiorze **makefile** lub w lini wywołania programu **make**. Makrodefinicja umieszczona w lini wywołania **make** przykrywa makrodefinicję znajdującą się w zbiorze **makefile**. Celem zwiększenia elastyczności programu **make**, w zbiorze **makefile** można zdefiniować specjalne makrodefinicje. Make używa ich jako domyślnych parametrów podczas generowania poleceń systemowych lub poszukiwania niewyspecyfikowanych zbiorów. Na przykład, jeśli dla zbioru **program.r** nie został wyspecyfikowany zbiór źródłowy, **make** poszuka zbioru **program.a** (lub .c, .f, .p) w katalogu określonym przez **SDIR** lub w bieżącym katalogu danych.

Make rozpoznaje następujące specjalne makrodefinicje:

ODIR=<path> Make poszukuje zbiorów o nazwach bez rozszerzenia w katalogu <path>. Jeśli ODIR nie jest zdefiniowane w zbiorze **makefile**, przeszukiwany jest katalog bieżący.

SDIR = <path> Make poszukuje zbiorów źródłowych, dla których nie podano pełnej ścieżki dostępu, w katalogu <path>. Jeśli SDIR nie jest zdefiniowane, przeszukiwany jest katalog bieżący.

RDIR=<path> Make poszukuje zbiorów relokowalnych, dla których nie podano pełnej ścieżki dostępu, w katalogu <path>. Jeśli RDIR nie jest zdefiniowane, przeszukiwany jest katalog bieżący.

CFLAGS=<opts> Powyższe opcje kompilacji używane są w poleceniach wywołania kompilatora.

RFLAGS=<opts> Powyższe opcje assemblacji używane są w poleceniach wywołania kompilatora.

LFLAGS=<opts> Powyższe opcje linkowania używane są w poleceniach wywołania kompilatora.

cc=<comp> Make używa kompilatora <comp>. Standardowym jest cc.

RC=<asm> MAKE używa assemblera <asm>. Standardowym jest r68.

Niektóre makrodefinicje rozwijane są przy wykonywaniu lini polecenia systemowego. W praktyce mają one charakter "wildcards" o następującym znaczeniu:

$@ Rozwinięcie do nazwy zbioru generowanego przez polecenie systemowe.

$* Rozwinięcie do przedrostka nazwy generowanego zbioru.

$? Rozwinięcie do listy nazw zbiorów, które zostały rozpoznane jako młodsze, niż zbiór docelowy danej lini zbioru **makefile**.

3.5.5 Polecenia systemowe generowane przez program make.

Make jest w stanie generować trzy rodzaje poleceń systemowych: wywołania kompilatora, assemblera i linkera.

1. Wywołanie kompilatora generowane jest, jeśli zbiór źródłowy o nazwie z rozszerzeniem .c, .p lub .f wymaga kompilacji. Wywołanie ma następującą postać:

```
$(cc) $(CFLAGS) -r=$(RDIR) $(SDIR)/<file>[.c, .f lub .p]
```

2. Wywołanie assemblera generowane jest, jeśli źródłowy zbiór asssemblerowy wymaga assemblacji. Wywołanie ma postać:

```
$(RC) $(RFLAGS) $(SDIR)/<file>.a -o=$(RDIR)/<file>.r
```

3. Wywołanie linkera generowane jest, jeśli wymagane jest linkowanie modułu ładowalnego. Wywołanie ma postać:

```
$(LC) $(LFLAGS) $(RELS)/<file>.r -f=$(ODIR)/<file>
```

Uwaga. Kiedy make generuje wywołanie linkera, przegląda swoją listę zbiorów i przekazuje jedynie pierwszy napotkany zbiór relokowalny.

Przykład:

```
prog: x.r u.r z.r
```

Wygenerowane zostanie polecenie cc x.r, nie zaś cc x.r y.r z.r,ani tez cc prog.r.

3.5.6 Opcje programu make.

Opcje programu można przekazywać w lini wywołania programu lub poprzez umieszczenie ich w zbiorze **makefile**.

?	Wyprowadza opis użycia programu **make**.
-b	Wyłącza posługiwanie się wbudowanymi regułami rządzącymi domyślnymi zależnościami między zbiorami. Program opierać się będzie jedynie o opis zależności umieszczony wprost w zbiorze **makefile**.
-bo	Wyłącza posługiwanie się wbudowanymi regułami domyślnych zależności zbiorów w odniesieniu do modułów ładowalnych.
-d	Opcja debagowania. Generowany jest listing makrodefinicji, nazw zbiorów z list zależności i daty ich modyfikacji.
-dd	Opcja intensywnego debagowania (bardzo rozwlekła).
-f-	Wczytuje zbiór **makefile** ze standardowego strumienia

wejścia.

-f=<path> <path> określa zbiór makefile. Jeśli <path> jest znakiem (-), polecenia meke wczytywane są ze standardowego strumienia wejścia.

-i Ignoruje błędy wykonania poleceń systemowych. Błąd wykonania polecenia ignorowany jest też, jeśli linia wywołania polecenia rozpoczęta jest znakiem (-). Normalnie make zatrzymuje się po wykryciu błędu.

-n Wyświetlanie poleceń systemowych bez wykonywania.

-s Wykonywanie poleceń systemowych bez wyprowadzania echa.

-t Uaktualnienie daty zbiorów bez wykonania poleceń systemowych.

-m Wykonanie poleceń systemowych niezależnie od dat zbiorów.

-x Użycie skrośnego kompilatora/assemblera.

-z=<path> Przekazanie listy zbiorów docelowych.

3.5.7 Przykłady użycia programu make.

Przykład 1.

Kompilacja programu C z określeniem wszystkich zależności między zbiorami i wszystkich poleceń systemowych:

```
program: xxx.r yyy.r
        cc xxx.r yyy.r -xf=program
xxx.r: xxx.c /d0/defs/oskdefs.h
        cc xxx.c -r
yyy.r: yyy.c /d0/defs/oskdefs.h
        cc yyy.c -r
```

Powyższy makefile określa, że program tworzony jest z dwóch zbiorow .r: xxx.r i yyy.r. Zbiory te są uzależnione odpowiednio od xxx.c i yyy.c, oba zaś od zbioru oskdefs.h. Jeśli xxx.c lub /d0/defs/oskdefs.h ma późniejszą datę niż xxx.r, wykonywane jest polecenie cc xxx.r -r. Jeśli yyy.c lub /d0/defs/oskdefs.h ma późniejszą datę niż yyy.r, wykonywane jest polecenie cc yyy.c -r. Je-

śli wykonywane jest chociaż jedno z tych poleceń, wykonywane jest polecenie **cc xxx.r yyy.r -xf=program.**

Ten sam przykład z wykorzystaniem wbudowanego mechanizmu samodzielnego generowania poleceń systemowych przez **make:**

```
program: xxx.r yyy.r
        cc xxx.r yyy.r -xf=program
xxx.r yyy.r:/d0/defs/oskdefs
```

Make przegląda katalog określony przez makrodefinicję **SDIR=<path>** i zestawia stosownie listę zależności. Standardowo przegląda katalog bieżący. Samoistnie generuje polecenie kompilacji **xxx.r** lub **yyy.r**, jeśli któryś ze zbiorów wymaga uaktualnienia.

Dalsze uproszczenie zbioru **makefile** byłoby możliwe, gdyby program był tworzony w oparciu o pojedyńczy zbiór źródłowy:

```
program:
```

Make dokona w tym przypadku następujących założeń:

- zbiór **program** jest modułem ładowalnym, uzależniony jest więc od zbiorów relokowalnych,
- nie załączono listy zależności, więc **make** zestawi w tablicy zależności powiązanie ze zbiorem **program.r**,
- **make** zestawi powiązanie zbioru źródłowego ze zbiorem relokowalnym.

Zakładając, że **make** znajdzie zbiór **program.a**, po sprawdzeniu dat poszczególnych zbiorów wygeneruje potrzebne wywołania spośród następujących poleceń systemowych:

```
r68 program.a -o=program.r
cc program.r -f=program.
```

Przykład 2:

Wewnętrzne własności programu **make** szczególnie są pomocne przy sprawdzaniu kilku modułów ładowania:

```
* początek
ODIR = /d0/cmds
RDIR = rels
UTILS = attr copy load backup dsave
SDIR = ../utils/source
utils.fils = $(UTILS)
```

```
touch utils.files
```

```
* koniec
```

Program make przegląda w katalogu rels zbiory attr.r, copy.r, etc. oraz w katalogu ../util/source zbiory attr.r, copy.c, etc. ; generuje odpowiednie polecenia systemowe dla wykonania kompilacji lub/i linkowania, jeśli zachodzi taka ppotrzeba. Jeśli jakikolwiek zbiór jest zakładany w katalogu UTIL, wykonywane jest polecenie touch utils.file odświeżające aktualną datę.

Przykład 3:

Makefile do tworzenia programu make:

```
* początek
ODIR = /h0/cmds
RDIR = rels
CFILES = domake.c doname.c dodate.c domac.c
RFILES = domake.r doname.r dodate.r
PFLAGS = -p64 -nh1
R2 = ../test/domac.r
RFLAGS = -q
make: $(RFILES) $(R2) getfd.r
    linker
$(RFILES): defs.h
$(R2): defs.h
    cc $*.c -r=../test
print.file: $(CFILES)
    pr $? $(PFLAGS) >-/p1
    touch print.files
* koniec
```

Przedstawiony makefile powoduje przeglądanie zbiorów z rozszerzeniem nazwy .r z listy makrodefinicji RFILES w katalogu rels. zaś zbioru domac.r w katalogu ../test (podano pełną sciezkę dostępu). Chociaż dla zbioru getfd nie podano wprost zależności, sprawdzane jest uzależnienie od zbioru getfd.a. Zbiory źródłowe poszukiwane są w katalogu bieżącym. Zbiór makefile może być używany do produkowania listingu. Wywołując polecenie make print.file, make rozwinie makrodefinicję %? w listę nazw zbiorów uaktualnionych pó-

źniej, niż został założony print.file. Jeśli print.file nie był założony, wylistowane będą wszystkie zbiory.

4. Komunikacja systemu OS-9 z otoczeniem

4.1 Proces wymiany danych pomiędzy PEP MC a terminalami graficznymi PC.

W komputerze PEP (w konfiguracji posiadanej w PIAP) z systemem OS-9 możliwe jest jednoczesne dołączenie czterech terminali (np. komputerów PC) realizujących niezależne funkcje komunikacyjne pomiędzy obsługa, a jednostką centralną w trakcie pracy na rzeczywistym obiekcie. Jeśli zaszłaby konieczność zwiększenia liczby terminali, można to uzyskać przez dołączenie do systemu dodatkowego pakietu 2xRS232. Pozwala to w praktyce na umieszczenie tych terminali na odległych stanowiskach w przedsiębiorstwach produkcyjnych, dzięki łączeniu ich przez interfejsy szeregowe. Typowy zestaw wyglądać może następująco: dwa PC-ty umieszczone w punkcie obserwacji - dyspozytorni, jeden w narzędziowni oraz jeden na we/wy systemu, gdzie wprowadza się detale do procesów obróbczych, podłączone oczywiście do MC PEP. Zadaniem terminali w dyspozytorni byłoby, niezależne od siebie, monitorowanie stanu systemu, modyfikacji parametrów procesu oraz ostrzeganie i informowanie o ewentualnych stanach nieprawidłowych zaistniałych w systemie. Dla prowadzenia nadzoru złożonego systemu automatyki niezbędny jest nowoczesny system monitoringu zapewniający dynamiczne obrazowanie stanu systemu w sposób jak najbardziej zbliżony do rzeczywistości. Dla zapewnienia wybiórczej obserwacji fragmentu systemu przygotowywana jest tzw. technika "zooming'u", wyjaśniona w dalszej części sprawozdania.

Możliwy jest także do natychmiastowego odczytu zestaw danych dotyczących wybranego elementu systemu. Dzięki współbieżności procesów programowych w systemie OS-9 możliwa jest modyfikacja dowolnego z zadań, przy zachowaniu właściwego sposobu współpracy z innymi zadaniami, bezpośrednio przez obsługę terminala w trakcie nieprzerwanej pracy całego systemu. Istotnym elementem takiego zestawu automatyki jest modułowość i niezależność zarówno oprogramo-

wania jak i sprzętu. Stąd też jako terminale będą proponowane mikrokomputery personalne PC, mogące przejąć wszystkie funkcje graficznej wizualizacji, wstępnej obróbki danych wprowadzanych do systemu sterowania oraz funkcje drukowania raportów sprawozdań itp. Komputery takie (połączone z minikomputerem PEP poprzez interfejs szeregowy) mogą być w dowolnym momencie odłączane od systemu sterowania pracującego w trybie ciągłym. Dzięki takiej niezależności sprzętowej i programowej zasadniczy program sterujący realizowany w komputerze PEP morze reagować niesłychanie szybko na każde zdarzenie zaistniałe podczas procesu wytwarzania. Takie rozwiązanie wsparte zastosowanym systemem operacyjnym OS-9, reagującym na zdarzenia, pozwala na realizacje dużych przedsięwzięć automatyki procesów produkcji. Dzięki narzędziu programowemu w MS-Windows terminal.exe możliwe jest wygodne uruchamianie oprogramowania komputera PEP spod oprogramowana aplikacyjnego Windows poprzez dwutorowe podłączenie wybranego terminala (PC), z zastosowaniem łączy RS-232, z komputerem PEP. Umożliwia to wykonywanie poleceń wewnątrz systemu OS-9, z jednoczesnym obserwowaniem zachowania się aplikacji, oraz, w wyniku potrzeby, jej modyfikacji.

4.2 Wizualizacja procesu sterowania za pośrednictwem terminala

Jako podstawę programową dla interpretacji graficznej procesów zachodzących podczas sterowania wybrano system MS-Windows. System ten zdobywa coraz większą popularność stając się standardem w technikach GUI (Graphics User Interface) co gwarantuje ciągłość rozwoju systemu oraz przyrost różnego rodzaju aplikacji i narzędzi programowych mogących w przyszłości znakomicie polepszać użyteczność systemu. Olbrzymią zaletą oprogramowania typu Windows jest to, ze umożliwia on okienkową wizualizację tworzonej aplikacji w sposób bardzo estetyczny z możliwością skalowania i przenoszenia w dowolne miejsce ekranu przy równoczesnym wyświetlaniu innych okien na tym samym ekranie. Operator systemu za pośrednictwem terminala wyposażonego w MS-Windows oraz odpowiednio przygotowaną aplikację może w jednym oknie ekranu oglądać rysunek interesującego go elementu hali fabrycznej wraz z dynamicznie kreślonymi czynnościami

dokonującymi się podczas tego procesu oraz w innych oknach ekranu jednocześnie obserwować wybrane fragmenty z inną dokładnością.

Oprogramowanie tego typu posiada uproszczoną formę wielozadaniowości, dzięki czemu możliwe jest korzystanie z przeróżnych funkcji jakie daje stosowanie mikrokomputera PC, z zachowaniem łączności w komunikowaniu się z komputerem głównym PEP. I tak np. operator może w jednym z okien przygotowywać określoną notatkę służbową, czy sprawozdanie, w innym oknie może zadać polecenie drukowania, a w oknie wspomnianej aplikacji (wizualizacja) wyświetlany będzie bierzący stan systemu. Każde z okienek systemu Windows może być zamienione w niewielki symboliczny rysunek (ikonę) bez przerywania procesów programowych wykonywanych w czasie pracy PEP. Dokładny i obszerny opis możliwości systemu Windows można znaleźć w wielu opracowaniach zarówno anglo-języcznych jak i Polskich. Warto tylko wspomnieć że licencja na użytkowanie systemu systemu MS-Windows jest w posiadaniu PIAP. Istnieją inne systemy realizujące podobne zadania na PC jak MS-Windows np QNX-Windows, X-Windows lecz ze względu na wielokrotnie wyższe ceny (MS-Windows 2,5mln, QNX-Windows+QNX ok 40mln, X-Windows+UNIX pow 70mln) oraz mniejszą dostępność na Polskim rynku komputerowym, zostały w niniejszych rozważaniach pominięte. Warto też wspomnieć, że Windows jest wygodnym narzędziem do tworzenia oprogramowania "końcowego" czyli programowych interfejsów z użytkownikiem, co może owocować skróceniem czasu realizacji zadań programowych w tym zakresie.

Tworzenie wspomnianej aplikacji w systemie Windows będzie realizowane w języku C++, jako że jest to najwygodniejsza forma dostosowania się do systemu Windows i jego zasobów bibliotecznych napisanych też w jęz C++. Ponadto stosowanie programowania obiektowego (C++) ułatwia pracę kilku programistom nad jednym problemem np. w przypadku tworzenia aplikacji do współpracy z MC PEP. Tworzone oprogramowanie w większym stopniu uniezależnia się wtedy od zmian personalnych osób tworzących oprogramowanie. Tworzenie aplikacji w systemie Windows wymaga ścisłego przestrzegania zbioru reguł określonych przez autorów tego systemu. Podstawowym zadaniem aplikacji obejmującej wizualizację pod systemem MS Windows jest stała obserwacja określonych zasobów pamięci komputera PEP i

41

przejmowanie zawartych tam informacji za pośrednictwem interfejsu szeregowego.

4.3 Proces wprowadzania danych technologicznych.

Przewidziane są trzy formy wprowadzania danych do systemu:

1. poprzez specjalne okienka informacyjne na monitorze terminala przygotowane do wprowadzania danych w odpowiednim formacie, wraz z opisem graficznym, do których niezbędne jest użycie klawiatury,
2. poprzez zestaw rozbudowywanego menu tekstowego (pull down menu)
3. poprzez wybór, za pomocą kursora myszy, jednego z symboli graficznych zestawionych w formie menu (np. wybór odpowiedniej technologii)
4. poprzez wybór i wystartowanie określonego zadania przedstawionego w postaci ikony przygotowanej uprzednio przez użytkownika, a realizującego pewien zestaw operacji wprowadzania i zmian parametrów.

Powyższe metody będą zastosowane do:

1. Wprowadzania danych dotyczących bieżącej, aktualnej technologii,
2. Wyboru, modyfikowania i wprowadzania specyficznych technologii obróbki kół zębatych,
3. Wprowadzania poleceń dotyczących zainstalowania określonych narzędzi,
4. Zestawiania dowolnych informacji o systemie, w stanie źródłowym lub po przetworzeniu.

5. Modyfikacja Założeń Techniczno-Ekonomicznych

5.1 Wstęp

W porównaniu z Założeniami Techniczno-Ekonomicznymi (spr. nr rej. 6636), odstępstwa od przewidywanych tam posunięć obejmują pkt. 4, pt.: "Przewidywany zakres prac modernizacyjnych". Wynika to z charakteru, przyjmowanego obecnie jako podstawa systemu, komputera PEP MC. Zaprezentowany powyżej system OS-9 (pkt. 2) istot-

nie różni się od systemu MS DOS przewidywanego uprzednio jako system operacyjny mający znaleźć zastosowanie przy oprogramowaniu fabryki FZ-200. Istotnie różni się również zestaw narzędzi programowych stosowanych do programowania (pkt.3). Wreszcie, zaproponowano nowe podejście przy komunikowaniu się pomiędzy PEP, a terminalem (pkt. 4). Na bazie tych przesłanek opracowano zbiór wytycznych, rozumianych jako modyfikacja ZTE.

5.2 Sprzęt

System OS-9 posadowiony będzie na niżej wyspecyfikowanym sprzęcie odmiennym od zestawionego w Założeniach Techniczno-Ekonomicznych (spr. nr rej. 6636):

Zestaw I: PEP MODULAR COMPUTERS.

1. Pakiet jednostki centralnej 68020 - VM20,
2. koprocesor zmiennoprzecinkowy 68882,
3. Hard Disk 130 MB VSCSI,
4. Floppy Disk 1.44 MB,
5. 3 x pakiet komunikacyjny VSIO,
6. System operacyjny OS/9-64k, a w tym:
   - jądro systemu,
   - kompilator C, assembler, linker, debuger,
   - biblioteka matematyczna,
   - edytor,
   - programy zarządzające.

Zestaw II: IBM PC 386/33 SVGA

1. 4 MB RAM,
2. Floppy Disk 1.2 MB 5.25",
3. Floppy Disk 1.44 MB 3.5",
4. Hard Disk 80 MB IDE,
5. Pamięć 16-Bit VGA w/512K,
6. Monitor kolorowy 14" 1024 x 768, 7. 3 x Parallel Port,
8. 2 x Serial Port,
9. 101-Key Keyboard,
10. Mouse INFO,
11. Pamięć Cache 64K, oraz
12. 2 x Drukarka firmy EPSON typ FX-1050 (15", 9 pin, 300 zn/s)

Zestaw III: IBBM PC 286/12 Hercules

1. 1 MB RAM,
2. Monitor monochromatyczny 14"
3. 1 x Serial Port,
4. Klawiatura przemysłowa,

Zestaw IV: IBM PC 286/12 Hercules

1. 1 MB RAM,
2. Monitor monochromatyczny 14"
3. 1 x Serial Port,
4. Klawiatura przemysłowa,

Funkcje tych komputerów w zmodernizowanym systemie, to:

I - sterowanie przepływem produkcji,
- kontrola ewidencji spływu produkcji,
- sterowanie pracą maszyn i urządzeń transportowych,
- tworzenie aktualnych baz danych,
- obsługa stanów alarmowych,
- analizowanie i archiwizowanie stanu obciążenia maszyn,
- rejestracja historii zdarzeń mających miejsce w systemie,

II - raportowanie na podstawie danych komputera I,
- wizualizacja stanów bieżących systemu,
- komunikacja operatorska z systemem.

III - podawanie do systemu numeru technologi oraz opisu wprowadzanej palety.

IV - zarządzanie paletami narzędziowymi,
- nadzór nad przebiegiem zmiany oprzyrządowania gniazd produkcyjnych,
- kontrola zgodności oprzyrządowania z wyselekcjonowaną technologią.

Wraz z wymienionym sprzętem komputerowym zostanie zakupiony system podtrzymywania pracy komputera po zaniku napięcia zasilania (UPS) o parametrach:

- czas pódtrzymania pracy komputerów - 5 min,
- moc zasilania przy napięciu 220 V~ - 500VA.

Wymiana sprzętu według powyższych zamierzeń wyeliminuje konieczność stosowania takich urządzeń peryferyjnych jak: zewnętrzne

pamięci bębnowe, dziurkarki i czytniki taśmy perforowanej, drukarki wierszowe i maszyny do pisania, usprawniając tym samym organizację sprzętową systemu, a zastosowanie zespołu podtrzymywania napięcia zasilania zabezpieczy przed utratą danych przetwarzanych w systemie w momencie pojawienia się ewentualnej awarii zasilania. Zostanie także wyeliminowane uciążliwe i zawodne codzienne uruchamianie systemu poprzez wczytywanie danych z taśm perforowanych, i zastąpione rozruchem automatycznym.

5.3 Oprogramowanie

Logika oprogramowania pozostaje bez zmian, natomiast istotną zmianą jest przyjęcie systemu OS-9 w miejsce systemu QNX.

Zakupiony system OS/9-64k będzie stanowił licencjonowaną własność Huty Stalowa Wola S.A.

W trakcie uruchamiania oprogramowania, nowo zestawiony sprzęt będzie dołaczany do systemu poprzez adapter umieszczony w dotychczasowej podstawie stanowiska komputera. Sygnały wyjściowe, BUS A, BUS DA, BUD DE, ST oraz BUS PU, będą doprowadzone do adaptera poprzez kable zakończone złaczami dopasowującymi komputer do wejścia adaptera. Kable te będą stanowiły integralną częsc zmodernizowanego sprzętu komputerowego.

6. Koszty przedsięwzięcia

Koszty przedsięwzięcia zostaną oszacowane podczas przygotowania dokumentów wymaganych do wystąpienia o projekt celowy, wspomagający modernizację fabryki FZ-200. Wystąpienie to jest obecnie konsultowane na szczeblu Zarządu Huty Stalowa Wola S.A. Wynika z niego, że wynagrodzenie dla PIAP sięgałoby 3.980 mln zł, natomiast wszelki wymagany sprzęt, którego koszty szacuje się na ok. 500 mln zł, zostałby zakupionu przez HSW S.A.