mgr inż. Wojciech NIKIEL
Przemysłowy Instytut Automatyki
i Pomiarów MERA–PIAP
Warszawa

PROBLEMY WYMIANY I PRZESYŁANIA INFORMACJI W SYSTEMACH CZASU RZECZYWISTEGO NA PRZYKŁADZIE ROZWIĄZAŃ ZASTOSOWANYCH W SYSTEMIE SIRTOS

W artykule przedstawiono różne algorytmy przesyłania informacji stosowane w systemie czasu rzeczywistego SIRTOS oraz rozpatrzono ich zalety i wady.

We wszystkich systemach R–T bardzo ważnym problemem jest wymiana informacji pomiędzy systemem a jego otoczeniem, jak również pomiędzy poszczególnymi zadaniami użytkowymi. Od efektywności rozwiązań programowych i sprzętowych tych problemów w dużym stopniu zależy sprawność działania całego systemu sterowania. W pracy rozpatrzono powyższe problemy i przedstawiono sposoby ich rozwiązania na przykładzie małego systemu czasu rzeczywistego SIRTOS opracowanego w Przemysłowym Instytucie Automatyki i Pomiarów MERA–PIAP w Warszawie, przeznaczonego dla przemysłowych sterowników mikroprocesorowych 16–bitowych.

Jednym z niezbędnych elementów systemu sterującego procesem technologicznym jest współpraca z operatorem pozwalająca na przyjmowanie z klawiatury poleceń i parametrów procesu oraz wyświetlanie na ekranie monitora aktualnych danych o przebiegu procesu. Zwykle ekran monitora i klawiatura współpracują z mikroprocesorem poprzez łącze szeregowe V–24. Łącze szeregowe ma wystarczającą przepustowość do spełniania tych funkcji. Najprostszym sposobem jest praca bezprzerwaniowa polegająca na wysyłaniu na ekran monitora kolejnych znaków z odpowiednią częstotliwością. Przy szybkości transmisji np. 4800 bodów, czas pomiędzy wysłaniem kolejnych znaków wynosi w przybliżeniu 2 ms, podczas których program musi czekać w „martwej pętli". Z punktu widzenia pracy wielozadaniowej i czasu wykorzystania procesora rozwiązanie takie jest nieefektywne. Można stosować je jedynie na etapie uruchamiania systemu albo w przypadku wykrycia poważnej awarii. Alternatywą jest praca przerwaniowa. Wysyłaniem znaków należy obarczyć jedno z zadań użytkowych. Po wysłaniu znaku na ekran zadanie to zawiesza się na semaforze w oczekiwaniu na przerwanie oznaczające gotowość do wysłania następnego znaku. Semafor jest podnoszony w procedurze obsługi tego przerwania. Powinien być to semafor z ogranicznikiem czasowym (ang. timeout). Możliwe jest wówczas wykrycie awarii polegającej na nieprzyjściu przerwania w ciągu określonego czasu (zależnego od szybkości transmisji) i podjęcie odpowiedniej obsługi sytuacji awaryjnej. Rozwiązanie to ma jednak również istotne wady, ponieważ wysłanie każdego znaku wymaga pracochłonnego przełączenia zadań użytkowych. Maleje zatem sprawność systemu sterowania S wyrażona wzorem:

$$S = \frac{T_u}{T_u + T_s} \cdot 100\%$$

gdzie:

$T_u = t_p + t_u + t_o$ jest czasem procesora wykorzystanym na pracę użytkową będącą sumą czasów:

t_p – czas przeznaczony na właściwą obsługę przerwań,

t_u – łączny czas wykonywania zadań użytkowych,

t_0 — czas „wolny" mikroprocesora (potencjalnie może zostać użyty na wykonanie zadań użytkowych).

T_s jest sumą czasu przeznaczonego na obsługę własną systemu — wykonania algorytmu koordynującego, przełączania zadań użytkowych, wykonania niezbędnych czynności związanych z obsługą przerwań (zachowania rejestrów, wysłania odpowiednich sygnałów do sterowników przerwań, odtworzenia rejestrów). W skrajnym przypadku, przy dużej prędkości transmisji, system może zajmować się tylko obsługą przerwania, wysłaniem nowego znaku i bezproduktywnym przełączaniem zadań. Zaletą tej metody jest natomiast możliwość natychmiastowego wykrycia opisanej powyżej sytuacji awaryjnej. Modyfikując nieco przedstawiony algorytm można uniknąć tego mankamentu. Zadanie użytkowe obsługującego ekran monitora powinno jedynie inicjować wysłanie komunikatu poprzez wysłanie pierwszego znaku, a następnie zawiesić się na semaforze w oczekiwaniu na zakończenie transmisji. Dalsze wysyłanie komunikatu — znak po znaku powinna przejąć procedura obsługi przerwania. Powrót z niej, po wysłaniu kolejnego znaku, następuje bezpośrednio do przerwanego zadania użytkowego, bez konieczności przełączania zadań i uruchamiania algorytmu koordynującego. Po wykryciu końca komunikatu, obsługa przerwania podnosi semafor odwieszając zadanie użytkowe obsługujące ekran monitora. W razie potrzeby inicjuje ono wysłanie następnego komunikatu. Awarie, polegającą na nieprzyjściu przerwania, można wykryć dopiero po upływie czasu przeznaczonego na cały komunikat (przy wykorzystaniu tego samego mechanizmu jak poprzednio). Każdorazowe obliczenie długości, a zatem i czasu wysyłania komunikatu, zwiększa nakład obliczeń. W niektórych przypadkach wystarczy ograniczyć tę wielkość z góry (np. przyjmując 120% czasu wysyłania najdłuższego komunikatu). W przypadku obsługi ekranu monitora dokładność tego oszacowania nie ma większego znaczenia ze względu na bardzo długi czas reakcji operatora systemu. Przełączanie zadań przy zastosowaniu tego algorytmu jest więc znacznie rzadsze — raz na komunikat, podczas gdy poprzednio — raz na znak (komunikaty wysyłane na monitor liczą zwykle od kilkunastu do kilkuset znaków). Opisany algorytm można wykorzystać również do transmisji danych na inne urządzenia zewnętrzne np.: drukarkę, dysk, dyskietkę, inny sterownik mikroprocesorowy itp. Algorytm nie zależy również od realizacji połączenia (transmisja szeregowa albo równoległa). System operacyjny musi jednak mieć rozwiązania programowe umożliwiające powrót bezpośrednio do przerwanego zadania, albo uruchomienie algorytmu koordynującego, w zależności od decyzji podjętej w obsłudze przerwania.

Inne problemy stwarza konieczność obsługi klawiatury operatora. I w tym przypadku możliwa jest praca bezprzerwaniowa. Wymaga ona odczytywania z odpowiednią częstotliwością (zwykle dużo mniejszą niż w przypadku transmisji na ekran) klawiatury przez program. Rozwiązanie to stosuje się tylko wyjątkowo. Naturalnym trybem pracy i w tym przypadku jest praca przerwaniowa. Po przyjściu przerwania, procedura obsługi odczytuje znak z klawiatury i wpisuje do bufora, uruchamiając jednocześnie zadanie użytkowe obsługi klawiatury. Zadanie to pobiera przesłany znak i przeprowadza jego identyfikację. Znaki „zwykłe" gromadzone są w buforze i tworzą komunikat, natomiast znaki „specjalne" traktowane są jako sterujące (np. znak „CR" kończy kompletowanie komunikatu, rozpoczyna jego interpretację i powoduje przesłanie do odpowiedniego zadania użytkowego). Ponadto zadanie obsługi klawiatury po przeprowadzeniu identyfikacji powinno wysyłać echo pobranego znaku na ekran monitora i przesunąć kursor. Czasami konieczna jest praca tego zadania bez wysłania echa, albo bez gromadzenia znaków w komunikat. Taki tryb pracy powinno zapewnić ustawienie odpowiednich parametrów przez użytkownika. Ze względu na znacznie mniejszą częstotliwość przerwań z klawiatury (zależną od sprawności operatora i nie przekraczającą w krótkich przedziałach czasu kilkuset znaków na minutę), nie występuje tutaj problem zbyt częstego uruchamiania zadania obsługi klawiatury.

Inaczej wygląda problem czytania informacji z dysku i dyskietek. W trakcie czytania przeprowadzane

jest (czasami wielokrotnie) pozycjonowanie głowicy, czytane są z dużą szybkością duże porcje informacji. Podobnie jest również w przypadku otrzymywania informacji z innego systemu mikrokomputerowego. W tych przypadkach należy zastosować analogiczny algorytm jak przy wypisywyniu komunikatów na ekran monitora. Kolejne bajty informacji powinny być gromadzone w buforze roboczym przez odpowiednią procedurę obsługi przerwań. Dopiero po zakończeniu otrzymywanej informacji, albo po zapełnieniu bufora, powinno zostać uruchomione zadanie przetwarzające ją dalej. W przypadku braku możliwości wstrzymywania dopływu informacji, albo przy konieczności ciągłego przetwarzania, napływające informacje należy gromadzić na przemian w dwóch buforach. Podczas analizy jednego bufora przez uzadanie użytkowe, napływające informacje powinny być gromadzone w drugim.

Zupełnie inne problemy występują przy konieczności wymiany informacji „wewnątrz systemu" pomiędzy zadaniami użytkowymi albo obsługą przerwań i zadaniem użytkowym. W systemie SIRTOS wykorzystano do tego celu mechanizm skrzynek pocztowych (ang. mailbox). Do ich realizacji zastosowano algorytm obsługi buforów cyklicznych realizowany przez funkcje systemowe. Bufor cykliczny jest wektorem o określonej długości i związanymi z nim trzema wskaźnikami. Dwa z nich określają pierwsze wolne miejsce w buforze i miejsce zajmowane przez najstarszą informację. Nowa informacja wpisywana jest zawsze na pierwsze wolne miejsce, natomiast w pierwszej kolejności pobierana jest informacja najstarsza. W ten sposób realizowana jest kolejka typu FIFO (ang. First In First Out). Jeżeli wpisywana informacja zapełni ostatnie miejsce w wektorze, następna wpisywana jest znowu na początek (stąd nazwa — bufor cykliczny). Oczywiście przy obu operacjach — pisania i czytania, modyfikowane są odpowiednie wskaźniki. Z wartości, jakie przyjmują oba te wskaźniki, nie wynika jeszcze jednoznacznie stan skrzynki pocztowej (zależny od „historii"). Trzeci wskaźnik jest trójwartościowym semaforem opisującym jej stan. Skrzynka pocztowa może znajdować się w stanie „pustym" (ang. EMPTY) — aktywnym tylko dla operacji pisania, „dostępnym" (ang. ACCESS) — możliwe są operacje pisania i czytania oraz „pełnym" (ang. FULL) — możliwa jest tylko operacja czytania. Oczywiście system operacyjny misu zapewnić odpowiednie mechanizmy obsługujące tak zbudowaną skrzynkę pocztową.

Następnym zagadnieniem jest postać informacji przekazywanych za pomocą skrzynek pocztowych. Jednym z możliwych rozwiązań jest przepisywanie całej informacji, wraz ze znacznikiem końca, do bufora — za pomocą funkcji systemowych. Wadą tego rozwiązania jest konieczność tworzenia bardzo dużych buforów i długi czas przepisywania informacji. Operacja taka musi przebiegać w sposób nieprzerywalny, a więc przy zamkniętym układzie przerwań. Zaletą jest natomiast szybkie zwolnienie pola roboczego w zadaniu wysyłającym informację. Nie równoważy to jednak dużych kosztów związanych z taką organizacją bufora. W systemie występują jednak sporadycznie przypadki, gdy takie rozwiązanie jest właściwe np. przy czytaniu informacji z klawiatury czytane są kolejne bajty (znaki) i zastosowanie powyższego algorytmu jest naturalne. Ze względu na zawsze jednakową porcję odczytywanej informacji (bajt), nie jest konieczne wpisywanie w tym przypadku znaczników końca.

Innym rozwiązaniem jest utworzenie bufora adresów. Zadania użytkowe przygotowują informację do wysłania we własnych polach roboczych i za pomocą funkcji systemowych przekazują do skrzynki pocztowej jedynie jej adres. Operacja taka jest bardzo szybka i nie wymaga dużej pamięci na utworzenie skrzynki pocztowej. W systemie może więc istnieć wiele takich obiektów. Parametry skrzynki pocztowej nie są również zależne od ilości przesyłanych informacji. Niedogodnością tego rozwiązania jest konieczność synchronizacji z konsumentem, w przypadku wielokrotnego wykorzystywania tego samego pola roboczego. W systemie SIRTOS nie stwarza to jednak większych problemów. Pierwszy bajt przesyłanej informacji można potraktować jako semafor ustawiany przez producenta i podnoszony po wykorzystaniu informacji przez konsumenta. Ponieważ semaforem może być dowolny bajt a

liczba semaforów w systemie nie jest ograniczona, rozwiązanie takie jest bardzo wygodne. W przypadkach, gdy synchronizacja z konsumentem jest niewskazana, należy używać wielu pól roboczych.

Ze względu na sposób obsługi zadań użytkowych przez system operacyjny, operacje wysyłania i odczytu informacji ze skrzynki pocztowej można podzielić na dwie grupy: przerwaniowe i bezprzerwaniowe. W pierwszym przypadku – zawsze po zapisie do pustej skrzynki pocztowej, albo po odczycie z pełnej, system operacyjny przegląda kolejkę zadań oczekujących na możliwość wykonania operacji na buforze. Możliwe jest zatem „przerwanie" bieżącego zadania pod wpływem zmiany stanu bufora i uruchomienie zadania o wyższym priorytecie. W drugim przypadku, jeśli tylko operacja na buforze jest możliwa do wykonania, nie powoduje ona ingerencji systemu operacyjnego. Jeżeli operacji na buforze nie można, wykonać, tzn. przy próbie czytania z pustego bufora, bądź zapisu do pełnego bufora, w obu powyższych grupach możliwe są dwa rozwiązania. W pierwszym – system operacyjny zawiesza zadanie użytkowe, aż do chwili, gdy żądana operacja będzie możliwa i automatycznie uruchamia je z powtórzeniem tej operacji. W drugim rozwiązaniu – zadanie informowane jest o wyniku operacji (pomyślnym albo nie) i do niego należy odpowiednia obsługa. Oba mechanizmy są niezbędne. Za pomocą pierwszego można organizować współpracę zadań typu producent – konsument, drugi natomiast jest niezbędny przy obsłudze sytuacji awaryjnych oraz przy współpracy procedur obsługi przerwań z zadaniami użytkowymi. Procedury te nie są oczywiście zadaniami i nie można ich „zawiesić", gdy żądana przez nie operacja na buforze nie jest możliwa:

Z przedstawionych powyżej rozważań wynika, że do problemu wymiany informacji w systemach czasu rzeczywistego należy przywiązywać dużą wagę. Sięgają one zazwyczaj bardzo głębokich warstw systemu i niejednokrotnie decydują o jego efektywności. Z tego też powodu stosowane rozwiązania oparte są na ścisłej współpracy zadań użytkowych z procedurami obsługi przerwań. Wynika z tego konieczność zwiększenia niezawodności sprzętu, kompatybilności elektromagnetycznej, jak również stosowania efektywnych metod programowych wykrywania awarii. Stosowanie skomplikowanych algorytmów wymaga użycia silnych narzędzi programowych zarówno przy tworzeniu oprogramowania systemowego jak i użytkowego.

Literatura

[1] Nikiel W., Wółtański St.: SIRTOS – A Small Industrial Real Time Operating System for Microcomputers. [W]: IFIP/IFAC Working Conference on Hardware and Software for Real Time Process Control. Vol. 2 Preprints. Warszawa, maj 1988 [oraz]: Hardware and Software for Real Time Process Control. Amsterdam, North–Holland 1989.

[2] Strzałkowski P., Wółtański St.: Systemy operacyjne czasu rzeczywistego dla mikrokomputerów 16–bitowych – charakterystyka porównawcza. Biuletyn PIAP nr 3/88.

[3] Deitel H. M.: An Introduction to Operating Systems. Addison–Wesley Publ. Comp., 1984. World Student Series.