ZESZYTY NAUKOWE POLITECHNIKI RZESZOWSKIEJ 288, Mechanika 85
RUTMech, t. XXX, z. 85 (4/13), październik-grudzień 2013, s. 515-524

Sławomir SAMOLEJ[1]
Tomasz ROGALSKI[2]
Dariusz NOWAK[3]

PROBLEMY IMPLEMENTACJI SYSTEMU STEROWANIA LOTEM NA PLATFORMĘ SYSTEMU OPERACYJNEGO CZASU RZECZYWISTEGO VxWorks

Wprowadzanie technik wytwarzania oprogramowania sterującego dla statków powietrznych oparte na systemach operacyjnych napotyka na pewne bariery psychologiczne. Procedury komunikacyjne, zarządzające procesami oraz sterujące urządzeniami wejścia/wyjścia są dostarczane w postaci skompilowanych modułów programowych, co często rodzi wątpliwości co do jakości i przewidywalności otrzymanych gotowych modułów programowych. Oprogramowanie przygotowywane na platformę systemu operacyjnego czasu rzeczywistego ma zwykle strukturę współbieżną, składającą się z komunikujących się między sobą i otoczeniem potencjalnie równolegle wykonywanych zadań. Opracowywanie aplikacji współbieżnych ma opinię zadania trudnego i niosącego wiele zagrożeń, takich jak możliwość zakleszczenia czy zagłodzenia procesów obliczeniowych. W pracy omówiono praktyczne zagadnienia związane z opracowywaniem oprogramowania systemu sterowania lotem na platformę wielozadaniowego systemu operacyjnego czasu rzeczywistego. Przeprowadzono dyskusję nad celowością stosowania systemów operacyjnych w aplikacjach awionicznych. Rozważano problemy dekompozycji podsystemów sterowania na zbiór współbieżnych zadań czasu rzeczywistego oraz zasady konstruowania kanałów komunikacyjnych pomiędzy komponentami systemu. Opracowanie podsumowuje doświadczenia zdobyte podczas wytwarzania oprogramowania autopilota dla Latającego Obserwatora Terenu, realizowanego jako projekt rozwojowy nr OR00011611.

Słowa kluczowe: statek powietrzny, system operacyjny czasu rzeczywistego, system sterowania lotem

[1] Autor do korespondencji/corresponding author: Sławomir Samolej, Politechnika Rzeszowska, al. Powstańców Warszawy 8, 35-959 Rzeszów, tel. (17) 8651477, e-mail: ssamolej@prz.edu.pl
[2] Tomasz Rogalski, e-mail: orakl@prz.edu.pl
[3] Dariusz Nowak, e-mail: darnow@prz.edu.pl

1. Wprowadzenie

Zasady konstruowania systemów sterowania bezzałogowych aparatów latających (BAL), jak się wydaje, stanowią już ugruntowaną dziedzinę wiedzy [1, 2], potwierdzoną aplikacjami zarówno na świecie [3, 4], jak i w Polsce [5, 6]. Na podstawie opracowanych rozwiązań następuje stopniowa rozbudowa oprogramowania platform latających i stacji naziemnych w celu poprawienia sterowania i zwiększenia spektrum zastosowań BAL [7-10, 11-13] lub zwiększenia bezpieczeństwa ich eksploatacji [14, 15]. W konsekwencji systematycznie wzrasta zapotrzebowanie na moc obliczeniową i zasoby komputerów pokładowych, a także powstaje problem doboru rozwiązań informatycznych pozwalających na ewolucję i rozbudowę oprogramowania tego typu urządzeń.

W pracy przedstawiono jedną z systematycznie rozwijanych ścieżek rozwoju oprogramowania systemów awionicznych – osadzanie aplikacji na platformach systemów operacyjnych czasu rzeczywistego. W kolejnych punktach zostanie podjęta dyskusja nad zasadnością takiego rozwiązania, a także zostaną przedstawione dodatkowe kluczowe decyzje projektowe, które należy podjąć podczas wytwarzania takiego rodzaju aplikacji. Podstawowym systemem operacyjnym analizowanym w pracy jest VxWorks [16-18].

2. Potrzeba wprowadzenia systemów operacyjnych – dyskusja

Wcześniejsze doświadczenia autorów [10, 12] obejmowały konstruowanie oprogramowania sterującego BAL na platformę mikroprocesorów jednoukładowych. Programy autopilotów osadzano na samodzielnie skonstruowanych systemach mikroprocesorowych, dla których opracowywano kompletne oprogramowanie zarządzające mikrokontrolerem i realizujące zadania sterowania lotem. Rozwiązanie takie ma niewątpliwe uzasadnienie z punktu widzenia kosztów, bezpieczeństwa i praktycznej możliwości bezpośredniego programowania peryferiów i zasobów systemu mikroprocesorowego. Platformę sprzętową mogą wtedy stanowić relatywnie tanie systemy uruchomieniowe lub wykonane niewielkim kosztem mikrokomputery. Dla niektórych mikrokontrolerów można uzyskać kompilatory za darmo lub w cenie nieprzekraczającej kilku tysięcy złotych. Kod programu w większości jest opracowywany przez programistę z wykorzystaniem standardowych bibliotek, co pozwala na bezpośredni wgląd we wszystkie zaprogramowane rozwiązania. W konsekwencji istnieje możliwość całkowitej analizy oprogramowania z punktu widzenia bezpieczeństwa.

Należy jednak zauważyć, że podejście takie do wytwarzania oprogramowania wymaga samodzielnego opracowania zarówno procedur zarządzających systemem mikroprocesorowym, jak i algorytmów sterujących, co wydłuża czas przygotowywania aplikacji. Pewien problem stanowi również utrzymanie i rozwój takiego rozwiązania informatycznego. W chwili gdy opracowany program należy przenieść na nowy system mikroprocesorowy, bardzo często pojawia się

konieczność ponownego opracowania warstwy oprogramowania odpowiedzialnej za wymianę informacji z systemem mikroprocesorowym (inne: architektura komputera, system obsługi przerwań, filozofia obsługi urządzeń wejścia/wyjścia). Kolejne utrudnienie może powodować włączanie nowej funkcjonalności do oprogramowania. Nowe procedury muszą zostać ściśle zintegrowane (skonsolidowane) z poprzednim oprogramowaniem. Pojawia się więc konieczność modyfikacji kodu źródłowego dotychczasowej aplikacji, co z kolei wymaga przeprowadzenia ponownej analizy całego oprogramowania. Współczesne mikroprocesory często oferują możliwość współbieżnego wykonywania programów. Samodzielne opracowywanie procedur zarządzających obliczeniami równoległymi na poziomie architektury mikroprocesora jest zagadnieniem trudnym i rzadko spotykanym poza zaawansowanymi naukowymi ośrodkami informatycznymi. Istotnym mankamentem bezpośredniego programowania mikrokontrolerów jednoukładowych jest również ograniczone spektrum interfejsów komunikacyjnych, które programista jest w stanie efektywnie oprogramować. Problemu nie stanowią tu standardowe interfejsy szeregowe czy CAN stosowane jako podstawowe magistrale komunikacyjne w mniejszych współczesnych aparatach latających. Wyzwaniem staje się jednak opracowanie obsługi interfejsu Ethernet/TCP/IP, który stopniowo jest wprowadzany do samolotów pasażerskich [19, 20], a w przyszłości ze względu na swoje zalety może się okazać dominującym standardem dla wszystkich statków powietrznych.

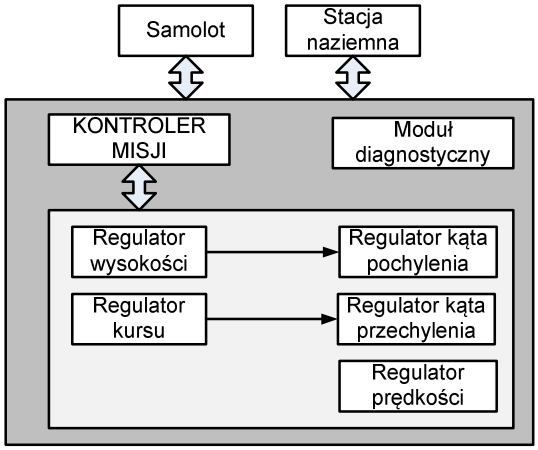
Rozwiązaniem, które w przypadku systematycznie rozwijanego oprogramowania ułatwia jego utrzymanie i rozbudowę, jest osadzenie aplikacji awionicznej na platformie systemu operacyjnego, a w przypadku aplikacji sterujących – specjalizowanego systemu operacyjnego czasu rzeczywistego. Zaletami tworzenia i utrzymania oprogramowania na platformę systemu operacyjnego są zwiększona przenoszalność, łatwość rozbudowy oraz mniejszy nakład pracy na przygotowanie aplikacji. Ponieważ oprogramowanie dla systemu operacyjnego jest opracowywane na swego rodzaju wirtualny procesor stworzony z usług systemu, w wielu wypadkach nie wymaga ono żadnych zmian podczas przenoszenia na nową platformę mikroprocesorowa pracującą pod kontrolą tego samego systemu operacyjnego. Włączenie nowej funkcjonalności w systemie można przeprowadzić przez dołączenie nowej usługi czy zadania, bez jakiejkolwiek ingerencji w kod źródłowy innych aplikacji. Podczas tworzenia aplikacji na system operacyjny czasu rzeczywistego dysponuje się zwykle zestawem sterowników dostarczonych wraz z systemem do zarządzania zasobami systemu mikroprocesorowego, a także mechanizmami umożliwiającymi tworzenie aplikacji wielozadaniowych z uwzględnieniem ograniczeń czasowych na wykonywanie pewnych obliczeń. Programowanie aplikacji sterujących na platformę systemu operacyjnego sprowadza się do opracowywania procedur realizujących prawa sterowania pogrupowanych w zbiory współpracujących zadań czasu rzeczywistego, posługujących się odpowiednimi wywołaniami usług systemowych do odwoływania się do urządzeń wejścia/wyjścia i komunikacji.

Opracowywanie oprogramowania na platformę systemu operacyjnego ma również wady wynikające z konieczności poniesienia większych kosztów oraz pokonania pewnych barier natury psychologicznej. Budowanie aplikacji na platformę profesjonalnych systemów operacyjnych czasu rzeczywistego (np. VxWorks) wymaga pokrycia kosztów systemu deweloperskiego do wytwarzania oprogramowania i skalowania systemu operacyjnego oraz zakupu bardziej zaawansowanego systemu mikroprocesorowego z przeinstalowanymi mechanizmami ładowania systemu i oprogramowania. W cenie docelowego produktu komercyjnego należy również uwzględnić opłaty licencyjne za zainstalowanie systemu operacyjnego. Za poniesione wydatki uzyskuje się bezpośrednią możliwość programowania współbieżnych aplikacji sterujących z dostarczonymi podsystemami zarządzającymi urządzeniami wejścia/wyjścia i magistralami komunikującymi (np. CAN, RS232, Ethernet/TCP/IP, USB). W ramach poniesionych kosztów otrzymuje się również niezwykle istotne na etapie tworzenia nowych aplikacji wsparcie techniczne od producenta systemu operacyjnego. System mikroprocesorowy, na którym ma zostać uruchomiona aplikacja sterująca BAL działająca pod nadzorem systemu operacyjnego czasu rzeczywistego, musi posiadać wydajny mikroprocesor oraz zwiększone zasoby pamięciowe, system operacyjny generuje bowiem dodatkowe obciążenie procesora. Wspomniane koszty można zredukować, posługując się bezpłatnymi wersjami systemów operacyjnych (np. LinuxRTAI [21] lub FreeRTOS [22]) lub tańszą wersją systemów profesjonalnych. W tym przypadku na programistę zostaje przeniesiony dodatkowy nakład pracy, ponieważ jest wtedy wymagane uzupełnienie systemu operacyjnego o autorskie sterowniki specjalizowanych urządzeń wejścia/wyjścia (np. CAN). Nie można również liczyć na wsparcie techniczne.

Wprowadzanie technik wytwarzania oprogramowania sterującego dla statków powietrznych oparte na systemach operacyjnych napotyka również na pewne bariery psychologiczne. Po pierwsze, w takich rozwiązaniach programista zwykle już nie dysponuje kompletnym kodem źródłowym swojej aplikacji. Procedury komunikacyjne zarządzające procesami oraz sterujące urządzeniami wejścia/wyjścia są dostarczone w postaci skompilowanych modułów programowych, do których nie ma się wglądu. Rodzi to często wątpliwości co do jakości i przewidywalności otrzymanych gotowych modułów programowych. Po drugie, oprogramowanie przygotowywane na platformę systemu operacyjnego czasu rzeczywistego ma zwykle strukturę współbieżną, składającą się z komunikujących się między sobą i otoczeniem potencjalnie równolegle wykonywanych zadań. Opracowywanie aplikacji współbieżnych ma opinię zadania trudnego i niosącego wiele zagrożeń, takich jak możliwość zakleszczenia czy zagłodzenia procesów obliczeniowych. W kolejnych rozdziałach pracy zostaną wskazane najważniejsze zagadnienia związane z techniką projektowania i programowania systemu sterowania BAL na platformę systemu operacyjnego czasu rzeczywistego VxWorks.

3. Dekompozycja systemu sterowania na zbiór zadań

Struktura systemu sterowania eksperymentalnego BAL została pokazana na rys. 1. System sterowania monitoruje odczyty czujników zainstalowanych na pokładzie samolotu oraz dane nadsyłane ze stacji naziemnej. Wyjścia regulatorów kąta pochylenia, kąta przechylenia oraz prędkości są powiązane z członami wykonawczymi. Informacja o stanie samolotu jest przesyłana do stacji. Kontroler misji na podstawie danych z czujników i stacji naziemnej identyfikuje aktualny stan samolotu i fazę odbywanej misji. Dokonuje również doboru rodzaju algorytmu nadrzędnego realizującego zadania regulacji wysokości i kursu oraz nadzoruje zadanie prowadzenia nawigacji. W systemie przewidziano osobny moduł diagnostyczny, który wykrywa uszkodzenia i rekonfiguruje system.

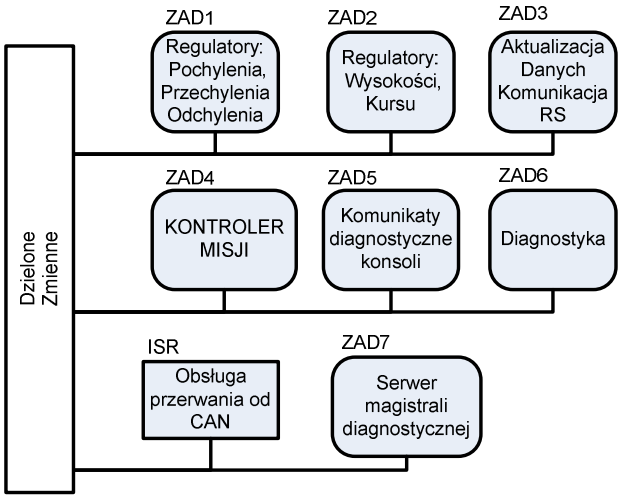


Rys. 1. Struktura systemu sterowania

Fig. 1. The structure of the control system

System sterowania eksperymentalnego BAL został zdekomponowany na osiem zadań czasu rzeczywistego (rys. 2.). Podstawowym kryterium dekompozycji jest częstotliwość, z jaką poszczególne algorytmy sterujące mają ponawiać swoje obliczenia. Co ustalony przedział czasowy, wynikający z właściwości dynamicznych sterowanego obiektu, następuje wznowienie obliczeń algorytmów sterowania zgrupowanych w danym zadaniu czasu rzeczywistego. Częstotliwość wykonywania obliczeń decyduje również o priorytecie zadań. Priorytety zadań zostały uszeregowane zgodnie z algorytmem Rate Monotonic [23], gdzie zadanie wymagające częstszego wykonywania otrzymują wyższe priorytety. Dysponując czasami wykonywania pojedynczego cyklu zadania i częstotliwościami wznawiania ich obliczeń, można analitycznie wykazać, czy ustalony zbiór zadań jest szeregowalny. Tabela 1. prezentuje cykle, oszacowane czasy obliczeń oraz priorytety poszczególnych zadań. Oszacowanie czasu wykonywania zadań dokonano z zastosowaniem odpowiednich funkcji systemowych (clock_gettime( )) VxWorks [16] oraz narzędzi do monitorowania pracy aplikacji VxWorks [17].

Analitycznie wykazano, że tak zaprojektowany zbiór zadań czasu rzeczywistego jest szeregowalny. Szczególnym zadaniem czasu rzeczywistego jest procedura obsługi przerwania od magistrali CAN. Jest to zadanie sporadyczne, które potraktowano jako zadanie cykliczne z pewną górną granicą częstości wznawiania obliczeń. Algorytmy sterujące zgrupowano w trzech zadaniach (ZAD1, ZAD2, ZAD4). Zadania ZAD3, ISR i ZAD7 odpowiadają za obsługę interfejsów komunikacyjnych, odpowiednio szeregowego (RS232), CAN i Ethernet/TCP/IP. Zadanie ZAD5 utrzymuje komunikację z komputerem monitorującym pracę systemu sterowania i umożliwia wymianę poleceń pomiędzy nim a komputerem sterującym. Zadanie ZAD6 realizuje algorytm diagnostyki sytemu. W systemie z pełną diagnostyką dopuszcza się koegzystencję trzech fizycznych sterowników, które komunikują się przez magistralę diagnostyczną (Ethernet/TCP/IP) i dokonują głosowania, który z nich przejmie sterowanie w przypadku wykrycia awarii jednego z nich. Interpretacja liczb oznaczająca priorytety zadań jest następująca: im niższa liczba, tym wyższy priorytet. Zadania są szeregowane zgodnie ze swoimi priorytetami z możliwością wywłaszczenia. Oznacza to, że jeśli w systemie pojawia się zadanie gotowe do wykonywania i ma ono wyższy priorytet niż zadanie aktualnie wykonywane, to następuje zawieszenie wykonywania zadania o niższym priorytecie i uruchomienie zadania o wyższym priorytecie.



Rys. 2. Struktura systemu sterowania

Fig. 2. The structure of the control system

Tabela 1. Parametry zadań czasu rzeczywistego

Table 1. Parameters of real-time tasks

Nazwa	Cykl [s]	Czas obliczeń [s]	Priorytet
ZAD1	0,02	0,004	120
ZAD2	0,10	0,001	160
ZAD3	0,10	0,004	170
ZAD4	0,10	0,002	180
ZAD5	1,00	0,002	210
ZAD6	0,02	0,001	130
ZAD7	0,10	0,001	220
ISR	0,02	0,001	50

4. Komunikacja

Podczas projektowania aplikacji współbieżnych czasu rzeczywistego należy zwrócić szczególną uwagę na dobór odpowiednich mechanizmów wymiany danych pomiędzy zadaniami. Przede wszystkim należy zapewnić dostarczanie bieżących danych o stanie systemu przy zapewnieniu ich spójności. Na rysunku 1. zasugerowano, że zadania czasu rzeczywistego systemu sterowania BAL współdzielą pewien obszar pamięci do wymiany danych. W modelu wymiany informacji pomiędzy poszczególnymi zadaniami przyjęto, że stan systemu będzie przechowywany we współdzielonych zmiennych, a zmienne będą przechowywać najbardziej aktualne informacje o systemie, tracąc „stare" dane. W konsekwencji procedury korzystające z danych dysponują zawsze najbardziej aktualnymi informacjami, przy czym dopuszcza się sytuację, gdy pewne dane zostaną utracone zanim kiedykolwiek zostaną skonsumowane. Spójność danych zachowuje się przez zaprogramowanie tzw. sekcji krytycznych na fragmenty procedur sterujących, w których następuje dostęp do współdzielonych zmiennych. Nie dopuszcza się do sytuacji, aby w czasie modyfikacji zmiennej przez jeden proces następowała równoczesna próba jej modyfikacji lub odczytu przez inny proces. W przypadku wymiany danych pomiędzy typowymi zadaniami sekcję krytyczną można zaprogramować z wykorzystaniem semaforów. Na rysunku 3. pokazano szkielet programu, w którym funkcja funcC zarówno w funkcji funcA, jak i funcB jest wykonywana w sekcji krytycznej. Oznacza to, że nie jest możliwe jej jednoczesne wykonywanie przez funkcje funcA() lub funcB(). Jeśli jest ona wykonywana w funkcji funcA(), to może być wykonywana w funcB() dopiero po jej zakończeniu w funkcji funcA() i zamknięciu sekcji krytycznej (semGive()). Funkcja semTake() przejmuje semafor (rozpoczyna sekcję krytyczną), funkcja semGive() – zwalnia go (kończy sekcję krytyczną). Jeśli semafor jest przejęty, to nie istnieje możliwość przejęcia go przez inny, współbieżnie wykonywany program.

```
// Inicjalizacja semafora…
funcA (){
    semTake (mySem, WAIT_FOREVER);
       funcC();// Sekcja krytyczna
    semGive (mySem);}
funcB (){
    semTake (mySem, WAIT_FOREVER);
       funcC();//Sekcja krytyczna
    semGive (mySem);}
```

Rys. 3. Schemat realizacji wzajemnego wykluczania z zastosowaniem semaforów

Fig. 3. The implementation scheme of mutual exclusion using semaphores

Pewnym wyzwaniem jest zachowanie zasady wzajemnego wykluczania dostępu do zmiennych, kiedy są one współdzielone pomiędzy typowymi zadaniami a procedurami obsługi przerwań. Jak się okazuje, zadania odwołujące się do takich współdzielonych zmiennych muszą się posłużyć zarówno mechanizmami blokującymi możliwość wywłaszczenia zadania (taskLock()/taskUnlock ()), jak i mechanizmami blokującymi wykonywanie przerwań (intLock()/intUnlock()).

Stosowanie mechanizmów zapewnienia wzajemnego wykluczania w dostępie do współdzielonych zmiennych wymaga szczególnie starannego programowania. Nieumiejętne posługiwanie się semaforami czy mechanizmami blokowania zadań lub przerwań może spowodować zagłodzenie lub zakleszczenie zadań albo zablokowanie systemu obsługi przerwań, a w konsekwencji awarię całego systemu sterowania. W najbardziej niekorzystnym zbiegu okoliczności zjawiska zakleszczenia mogą być niewykryte na etapie projektowania i testów systemu, a ujawnić się w czasie jego eksploatacji.

5. Podsumowanie

W pracy wskazano dodatkowe kluczowe etapy projektowania sterujących aplikacji awionicznych, przy założeniu, że będą one osadzane na systemach operacyjnych czasu rzeczywistego. Przedyskutowano również zasadność stosowania systemów operacyjnych jako platform dla aplikacji lotniczych. Wydaje się, że zastosowania systemów operacyjnych w aplikacjach awionicznych będą w najbliższym czasie istotną gałęzią rozwoju bezzałogowych i załogowych obiektów latających.

Literatura

[1] Austin R.: Unmanned aircraft systems. UAVS Design, Development and Deployment, John Wiley & Sons, 2010.

[2] Lozano R. (ed.): Unmanned aerial vehicles. John Wiley & Sons, 2010.

[3] http://www.avinc.com/uas/.

[4] http://www.ga-asi.com/products/aircraft/.

[5] http://www.eurotech.com.pl/produkty-i-uslugi/2/lotnictwo-i-awionika.

[6] http://www.wb.com.pl/Rozwiazania,Systemy-C4ISR,Systemy-rozpoznania.html.

[7] Bachuta M.J., Czyba R., Janusz W., Yurkevich V.D.: UAV glider control system based on dynamic contraction method. 17th Int. Conf. Methods and Models in Automation and Robotics (MMAR), 2012, pp. 114-118.

[8] Campoy P. et al.: Computer vision onboard UAVs for civilian tasks. J. Intelligent Robotic Systems, 54 (2008), 105-135.

[9] Gosiewski Z., Kulesza Z.: Application of unfalsified control theory in controlling MAV. Solid State Phenomena, 198 (2013), 171-175.

[10] Jaromi G., Rogalski T., Rzucidło P., Wałek Ł.: Integracja układu sterowania z mini BSL. V Konferencja Awioniki, Rzeszów 2007.

[11] Kopecki G., Pieniążek J., Rogalski T., Rzucidło P., Tomczyk A.: Proposal for navigation and control system for small UAV. Aviation 14 (2010), 77-82.

[12] Rogalski T.: The control algorithms for manoeuvring flying target. Scientific Aspects of Unmanned Mobile Vehicle, Politechnika Świętokrzyska, Kielce 2010, t. 1, 177-184.

[13] Zhou G., Zang D.: Civil UAV system for earth observation. Geoscience and Remote Sensing Symposium, IGARSS 2007, pp. 5319-5322.

[14] Dołęga B., Rzucidło P.: Monitorowanie pracy układu sterowania bezzałogowym aparatem latającym. ZN AMW, Gdynia 2011, t. LII, 83-91.

[15] Grzybowski P., Kordos D., Rzucidło P.: Metody zapewnienia bezpieczeństwa z poziomu naziemnej stacji kontroli lotu. Mat. konf. „Bezzałogowe Statki Powietrzne w Polsce", Warszawa 2012.

[16] VxWorks Kernel Programmer's Guide 6.8. Wind River Systems, Inc., 2009.

[17] Wind River System Viewer User's Guide 3.2. Wind River Systems, Inc., 2009.

[18] Wind River VxWorks Platforms User's Guide 3.8. Wind River Systems, Inc., 2009.

[19] AFDX: The next generation interconnect for avionics subsystems. Avionics Magazine Tech. Report, 2008.

[20] Samolej S., Rogalski T., Kopecki G., Tomczyk A.: The integration of a prototype pitch control application with IMA2G devices. Automatyka, Wydaw. Naukowo-Dydaktyczne AGH w Krakowie, artykuł zaakceptowany do druku.

[21] https://www.rtai.org/.

[22] http://www.freertos.org/.

[23] Klein M.H.: A practitioner's handbook for real-time analysis: Guide to rate monotonic analysis for real-time systems. Kluwer Academic Publishers, 1993.

FLIGHT CONTROL SYSTEM ON THE VxWorks REAL-TIME OPERATING SYSTEM PLATFORM – SELECTED IMPLEMENTATION ISSUES

Abstract

Implementation of manufacturing techniques of control software for aicrafts based on operating systems encounters psychological difficulties. Communication procedures of processes management and control input-output attachments are delivered as complicated programmatic modules which raise doubts about quality and predictability of ready-to-use programmatic modules. Real-

time operating system software has usually concurrent structure consisting of potentially parallel tasks communicating with each other. Developing of concurrent applications is a difficult task and is characterized by number of risks, such as possibility of deadlock or starvation of computational processes. Some new development steps ought to be taken into consideration during the real-time operating system based on flight control system programming. These steps as well as the advantages and disadvantages of real-time operating systems introduction in avionic applications are discussed in this paper. The technique of decomposition of flight control functions into a set of cooperating real-time tasks and inter-tasks data exchange strategies are also briefly presented. The research reported in this paper was supported by Polish scientific founds as a development project No. OR00011611.

Keywords: aircraft, real-time operating system, flight control system

DOI:10.7862/rm.2013.46

Otrzymano/received: 15.09.2013 r.
Zaakceptowano/accepted: 22.11.2013 r.