Systemy operacyjne czasu rzeczywistego

Szeregowanie zadań. Podsumowanie. Zagadnienia na kolokwium

Paweł Strączyński
pstraczynski@tu.kielce.pl
Katedra Urządzeń Elektrycznych i Automatyki

Wywłaszczające planowanie oparte na priorytetach

Domyślny planer w systemie Xenomai to wywłaszczające planowanie FIFO oparte na priorytetach.
W związku z tym wykonanie zadania o niskim priorytecie jest przerywane, gdy zadanie o wysokim priorytecie jest gotowe do wykonania.
Gdy wiele procesów o tym samym priorytecie jest gotowych do uruchomienia, domyślnie zadanie, które jako pierwsze zostało umieszczone w kolejce oczekiwania w harmonogramie, jest uruchamiane jako pierwsze (kolejność FIFO) i uruchamiane do końca.

Wywłaszczające planowanie oparte na priorytetach

W harmonogramie opartym na priorytetach wywłaszczania każde zadanie ma priorytet, a jądro zapewnia przydzielenie procesora do zadania o najwyższym priorytecie, które jest gotowe do uruchomienia.
Ta metoda planowania jest wywłaszczająca, ponieważ jeśli zadanie o wyższym priorytecie niż bieżące zadanie stanie się gotowe do uruchomienia, jądro zapisuje kontekst bieżących zadań i przełącza się na kontekst zadania o wyższym priorytecie.
Jądro Xenomai ma priorytety między najwyższym priorytetem 99 a najniższym priorytetem 1.
Podczas tworzenia zadania za pomocą `rt_task_create(..)` jednym z argumentów do jest priorytet, z jakim zadanie ma zostać wykonane. Priorytet zadania można zmienić za pomocą funkcji:

```
int rt_task_set_priority(RT_TASK *task, int prio)
```

Wywłaszczające planowanie oparte na priorytetach

```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <native/task.h>
#include < native /sem.h>
#include < native /timer.h>
#define NTASKS 3
#define HIGH 52 /* high priority */
#define MID 51 /* medium priority */
#define LOW 50 /* low priority */
RT_TASK demo_task[NTASKS];
RT_SEM mysync;
#define EXECTIME 2e8 // execution time in ns
#define SPINTIME 1e7 // spin time in ns
void demo(void *arg)
{
      RTIME starttime, runtime;
      RT_TASK_INFO taskinfo;
      int num=*(int *)arg;
      printf("Task : %d\n",num);
      rt_sem_p(&mysync,TM_INFINITE);
      runtime = 0;
      while(runtime < EXECTIME) {
            rt_timer_spin(SPINTIME); // spin cpu doing nothing
            runtime = runtime + SPINTIME;
            printf("Running Task : %d at ms : %d\n",num, runtime/1000000);
      }
      printf("End Task : %d\n",num);
}

//startup code
void startup()
{
      int i;
      char str[20] ;
      // semaphore to sync task startup on
      rt_sem_create(&mysync,"MySemaphore",0,S_FIFO);
      for(i=0; i < NTASKS; i++) {
            printf("start task : %d\n",i);
            sprintf(str,"task%d",i);
            rt_task_create(&demo_task[i], str, 0, 50, 0);
            rt_task_start(&demo_task[i], &demo, &i);
      }
      // assign priorities to tasks
      rt_task_set_priority(&demo_task[0],LOW);
      rt_task_set_priority(&demo_task[1],MID);
      rt_task_set_priority(&demo_task[2],HIGH);
      printf("wake up all tasks\n");
      rt_sem_broadcast(&mysync);
}

int main(int argc, char* argv[])
{
      startup();
      printf("\nType CTRL-C to end this program\n\n" );
      pause();
}
```


Źródło: https://www.cs.ru.nl/MSchuts/DES/XenomaiExercises/Exercise-4.html

Planowanie Round-Robin

W przypadku zadań o równym priorytecie program planujący Xenomai obsługuje zarówno zasady planowania FIFO, jak i algorytmu round-robin.

Algorytm szeregowania round-robin jest podobna do systemu operacyjnego z podziałem czasu, który uruchamia każdy aktywny proces po kolei przez swój udział w czasie ("przedział czasu").

W przypadku planowania FIFO zadanie wywłaszczy tylko zadanie o niższym priorytecie; nigdy nie przesądza zadania o takim samym lub niższym priorytecie.
Gdy wiele zadań o tym samym priorytecie jest gotowych do uruchomienia, ten, który został umieszczony jako pierwszy w kolejce oczekiwania harmonogramu, jest uruchamiany jako pierwszy (zgodnie z kolejnością FIFO). Dopiero gdy pierwsze zadanie jest gotowe, planista może wykonać drugie oczekujące zadanie z oczekującej kolejki.

Planowanie Round-Robin

Algorytm planowania round-robin próbuje osiągnąć sprawiedliwe planowanie wśród wszystkich gotowych zadań o tym samym priorytecie.

Bez planowania round-robin pojedyncze zadanie może przywłaszczyć sobie procesor, nigdy nie blokując się, a tym samym nigdy nie dając szansy na wykonanie innym zadaniom o równym priorytecie.

Algorytm ten zapewnia sprawiedliwy przydział procesora do zadań o tym samym priorytecie dzięki podejściu znanemu jako podział czasu (*slice*).
Każde zadanie jest wykonywane przez określony interwał lub przedział czasu. Następnie wykonuje się kolejne zadanie w równych odstępach czasu, naprzemiennie.

Przydział jest sprawiedliwy, ponieważ żadne zadanie z grupy priorytetowej nie otrzymuje drugiego wycinka czasu, zanim inne zadania grupy nie otrzymają wycinku.

Planowanie Round-Robin

Planowanie round-robin można włączyć wewnątrz zadania za pomocą wywołania funkcji

```
rt_task_set_mode(0,XNRRB,NULL);
```

Funkcja:

```
rt_task_slice(NULL, quantum);
```

Służy do ustawienia interwału wycinka dla planera round-robin.

Ten interwał to czas, przez jaki każde zadanie może być uruchomione przed przekazaniem procesora innemu zadaniu o równym priorytecie.

Te funkcje należy wywołać tylko raz.

Planowanie Round-Robin

```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <native/task.h>
#include <native/sem.h>
#include <native/timer.h>
#define NTASKS 3
RT_TASK demo_task[NTASKS];
RT_SEM mysync;
#define EXECTIME 2e8 // execution time in ns
#define SPINTIME 1e7 // spin time in ns
void demo(void *arg) {
    RTIME starttime, runtime;
    int num=*(int *)arg;
    printf("Task : %d\n",num);
    rt_sem_p(&mysync,TM_INFINITE);
    // let the task run RUNTIME ns in steps of SPINTIME ns
    runtime = 0;
    while(runtime < EXECTIME) {
        rt_timer_spin(SPINTIME); // spin cpu doing nothing
        runtime = runtime + SPINTIME;
        printf("Running Task : %d at time : %d\n",num,runtime);
    }
    printf("End Task : %d\n",num);
}

//startup code
void startup() {
    int i;
    char str[20] ;
    // semaphore to sync task startup on
    rt_sem_create(&mysync,"MySemaphore",0,S_FIFO);
    for(i=0; i < NTASKS; i++) {
        printf("start task : %d\n",i);
        sprintf(str,"task%d",i);
        rt_task_create(&demo_task[i], str, 0, 50, 0);
        rt_task_start(&demo_task[i], &demo, &i);
    }
    printf("wake up all tasks\n");
    rt_sem_broadcast(&mysync);
}

int main(int argc, char* argv[])
{
     startup();
     printf("\nType CTRL-C to end this program\n\n" );
     pause();
}
```

Zagadnienia na kolokwium

1. Co to jest system operacyjny czasu rzeczywistego?
2. Podział systemów czasu rzeczywistego. Scharakteryzuj krótko każdy rodzaj.
3. Wymień znane Ci systemy czasu rzeczywistego.
4. Co to jest RT_preempt patch? Opisz krótko czym się charakteryzuje.
5. Co to jest proces.
6. Czym jest zadanie czasu rzeczywistego.
7. Co to jest priorytet zadania w systemie operacyjnym czasu rzeczywistego
8. Co to są sygnały? Do czego służą? Opisz wybrany sygnał w SO.
9. Co to jest synchronizacja międzyprocesowa? Wymień znane Ci mechanizmy synchornizacji procesów.
10. Co to jest mutex? Przedstaw przykład działania mutex'u.
11. Co to jest semafor? Przedstaw przykład działania semafora.
12. Do czego służy komunikacja międzyprocesowa. Podaj przykład zastosowania. Wymień znane CI mechanizmy komunikacji międzyprocesowej.
13. Co to jest sekcja krytyczna? Jakie znasz metody blokowania dostępu do sekcji krytycznej?
14. Scharakteryzuj krótko komunikacje przez pamięć wspólną.
15. Scharakteryzuj krótko komunikacje przez kolejkę komunikatów.
16. Omówić algorytm szeregowania zadań w systemie Xenomai.