Programovací jazyk C
vlákna

Miroslav Melicherčík

Rozdelenie úlohy

- najjednoduchší spôsob rozdelenia úlohy pomocou príkazu fork()
- vytvorí nový nezávislý proces identický s pôvodným procesom
- navratová hodnota v pôvodnom procese je PID nového procesu
- navratová hodnota v novom procese je 0
- komunikácia medzi procesmi je možná pomocou rúr (pipe) – čaká kým nepríde hodnota
- problém, pokiaľ chceme zdieľať premenné

Príklad

```
pid = fork();
if(pid < 0) {
    fork_error_function();
} else if(pid == 0) {
    child_function();
} else {
    parent_function();
}
```

Príklad

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#define ITEMS 500000

int share, pid;
int pi[2];
```

Príklad pokračovanie

```
void konzument(void *a){
  printf("Process %s: start\n", (char *)a);
  while(1){
    read(pi[0], &share, sizeof(int));
    printf("Process %s: %i\n", (char *)a, share);
    if (share == -1){
      break;
    };
  };
  printf("Process %s: end\n", (char *)a);
  return((void)NULL);
}
```

Príklad pokračovanie

```
void producent(void *a){
  int i;
  printf("Process %s: start\n", (char *)a);
  for (i = 0; i<ITEMS; i++){
    share = (int)rand();
    if (share == -1) share = 0;
    if (i == ITEMS - 1) share = -1;
    printf("Process %s: %i\n", (char *)a, share);
    write(pi[1], &share, sizeof(int));
  };
  printf("Process %s: end\n", (char *)a);
  return((void)NULL);
}
```

Príklad pokračovanie

```
int main()
{
  pipe(pi);
  if (pid=fork(),pid==0) {
    close(pi[0]);
    producent((void*)"producent");
  } else {
    close(pi[1]);
    konzument((void*)"konzument");
  }
  return 0;
}
```

Vlákna

- inou možnosťou ako rozdeliť program na niekoľko podčastí je použitie vlákien (threads)

```
THREAD 1
a = data;
a++;
data = a;
```

```
THREAD 2
b = data;
b--;
data = b;
```

- problém však nastane keď chcem pracovať so zdieľanou premennou
- nie je zaručené, s akou hodnou data program skončí

Vlákna

- možnosti: ak bolo na začiatku data = 0
  data = -1
  data = 0
  data = 1
  data = iné číslo (ak bolo použité v dobe prepisu)

THREAD 1	THREAD 2
`a = data;`	`b = data;`
`a++;`	`b--;`
`data = a;`	`data = b;`

- potrebujem riadiť prístup k zdieľanej premennej

Vytvorenie vlákna

- pomocou príkazu z štandardnej knižnice pthread.h

```
#include <pthread.h>

int pthread_create (pthread_t *thread_id,
                    const pthread_attr_t *attributes,
                    void *(*thread_function)(void *),
                    void *arguments);
```

Ukončenie vlákna

- vlákno môžem explicitne ukončiť pomocou príkazu

```
int pthread_exit (void *status);
```

- v pôvodnom procese musím počkať, kým neskončia všetky vlákna
- ukončenie pôvodného procesu by viedlo k predčasnému ukončeniu vlákien
- alebo ak potrebujem aby iné vlákno čakalo na ukončenie ďalšieho

```
int pthread_join (pthread_t thread, void **status_ptr);
```

Riadenie prístupu

- mutex
- join
- semafór

Mutex

- zabezpečí vzájomné vylúčenie súčasného prístupu k zdieľanému prostriedku (zamkne kritickú oblasť)
- inicializácia mutexu

```
int pthread_mutex_init (pthread_mutex_t *mut,
const pthread_mutexattr_t *attr)
```

- zamknutie mutexu

```
int pthread_mutex_lock (pthread_mutex_t *mut)
```

- odomknutie mutexu

```
int pthread_mutex_unlock (pthread_mutex_t
*mut)
```

- zrušenie mutexu

```
int pthread_mutex_destroy (pthread_mutex_t
*mut);
```

Príklad (mutex + join)

```
#include <stdio.h>
#include <pthread.h>

#define NTHREADS 10
void *thread_function(void *);
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int counter = 0;

main() {
  pthread_t thread_id[NTHREADS];
  int i, j;

  for(i=0; i < NTHREADS; i++) {
    pthread_create( &thread_id[i], NULL, thread_function, NULL );
  }
  for(j=0; j < NTHREADS; j++) {
    pthread_join( thread_id[j], NULL);
  }
  printf("Final counter value: %d\n", counter);
}

void *thread_function(void *dummyPtr) {
  printf("Thread number %ld\n", pthread_self());
  pthread_mutex_lock( &mutex1 );
  counter++;
  pthread_mutex_unlock( &mutex1 );
}
```

Semafór

- slúži na riadenie prístupu k zdieľaným prostriedkom
- je potrebné použiť knižnicu semaphore.h
- inicializácia semaforu
  `int sem_init(sem_t *, int, unsigned)`
- nastavenie signálu semaforu
  `int sempost(sem_t *)`
- čakanie na signál voľno
  `int semwait(sem_t *)`
- zrušenie semaforu
  `int sem_destroy(sem_t *)`

Príklad

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#define ITEMS 99999

sem_t semfull;
sem_t semempty;
int share;
```

Príklad

```
void * konzument(void *a){
  printf("Process %s: start\n", (char *)a);
  while(1){
    sem_wait(&semfull);
    printf("Process %s: %i\n", (char *)a,
share);
    if (share == -1){
      sem_post(&semempty);
      break;
    };
    sem_post(&semempty);
  };
  printf("Process %s: end\n", (char *)a);
  return NULL;
}
```

Príklad

```
void * producent(void *a){
  int i;
  printf("Process %s: start\n", (char *)a);
  for (i = 0; i<ITEMS; i++){
    sem_wait(&semempty);
    share = (int)rand();
    if (share == -1) share = 0;
    if (i == ITEMS - 1) share = -1;
    printf("Process %s: %i\n", (char *)a,
share);
    sem_post(&semfull);
  };
  printf("Process %s: end\n", (char *)a);
  return NULL;
}
```

Príklad

```
int main(){
	pthread_t a,b;

	sem_init(&semfull, 0, 0);
	sem_init(&semempty, 0, 1);
	pthread_create(&a, NULL, producent, (void*)"producent");
	pthread_create(&b, NULL, konzument, (void*)"konzument");
	pthread_join(a, NULL);
	pthread_join(b, NULL);
	sem_destroy(&semfull);
	sem_destroy(&semempty);

	return 0;

}
```

Zdieľanie pamäte

- keď potrebujeme zdieľať určitú časť pamäte medzi viacerými procesmi
- procesy vytvoríme pomocou príkazu fork()
- vytvorenie linku na zdieľaný objekt

```
int shm_open(const char *name, int oflag, mode_t mode)

int shm_unlink(const char *name)
```

- namapovanie súboru alebo zariadenie do pamäte

```
void *mmap(void *start, size_t length, int prot, int flags, int
  fd, off_t offset)

int munmap(void *start, size_t length)
```

- alokácia segmentu zdieľanej pamäte

```
int shmget(key_t key, size_t size, int shmflg)
```