CSC552 – Advanced UNIX Programming

Semaphores

Dr. L. Frye
Kutztown University

Background

- Dijkstra – 1965
- Uses
  - Mutual exclusion
  - Synchronization
- Two operations
  - down, P, wait, lock, semaphore lock – sleep
  - up, V, signal, unlock, post, semaphore unlock – wakeup
- Atomic operation

Down/wait

```
void wait(semaphore_t *sp)  {
        if (sp->value > 0)
           sp->value--;
        else {
           /* add process t sp->list */
           /* block */
        }
}
```

Up/signal

```
void signal(semaphore_t *sp)  {
  if (sp->list != NULL)
     /* remove a process from sp->list and
                put in ready state */
  else
     sp->value++;
}
```

POSIX Semaphores

- POSIX:SEM
- Two types
  - Named
  - Unnamed
- Must initialize
  - sem_init()
- Destroy
  - sem_destroy()
- Close
  - sem_close()
  - sem_unlink()

Named Semaphores

- Similar to files
  - Name
  - User ID, group ID
  - Permissions
- Must open
  - sem_open()

Example

- semaphores/chaincritical.c
- semaphores/chainnamed.c

Semaphore Operations

- sem_post()
- sem_wait()
- sem_trywait()

Examples

- semaphores/semshared.c
- semaphores/threadcritical.c
- semaphores/threadcriticalsem.c

Semaphore Sets

- POSIX:XSI
- Array of semaphore elements
- Each element
  - semval
  - sempid
  - semncnt
  - semzcnt
- Data structure → semid_ds (page 515)
- Two queues
  - Waiting for value to equal 0
  - Waiting for value to increase

Semaphore Set Functions

- semget()
- semctl()
- removesem()
- semop()

- semaphores/chainsemset.c

Producer-Consumer Semaphores

```
#define  N    100     /* number of slots in buffer */
typedef  int  semaphore;
semaphore  mutex = 1; /* critical section control */
semaphore empty = N;  /* empty buffer slots */
semaphore full = 0;       /* full buffer slots */
```

```
void producer()
{
    int item;
    while (TRUE)  {           /* loop forever */
        produce_item(&empty); /* generate next item */
        down(&empty);      /* decrement empty count */
        down(&mutex);      /* enter critical section */
        enter_item();                /* put item in buffer */
        up(&mutex);           /* leave critical section */
        up(&full);     /* increment count of full slots */
     }
}   /* end producer */
```

```
void consumer()
{
   int item;

   while (TRUE)  {            /* loop forever */
      down(&full);            /* decrement full count */
      down(&mutex);           /* enter critical section */
      remove_item(&item);
      up(&mutex);             /* leave critical section */
      up(&empty);  /* increment empty slots count */
      consume_item(item);          /* do something */
   }
}   /* end consumer */
```

Explanation

- Binary semaphore
- Two uses for semaphores
  - Mutual exclusion
  - Synchronization
- What would happen if the two DOWN's in the producer code were reversed in order, so mutex was decremented before empty instead of after it?