IOWA STATE UNIVERSITY

Department of Electrical and Computer Engineering

Lecture 13:
Inter-Process Communication (IPC) III

Agenda

- Recap
- Inter-Process Communication III
    - Semaphore
    - Conditional Variables

Recap

- Solutions of Mutual Exclusion
  - Peterson's Solution

```
#define FALSE  0
#define TRUE   1
#define N      2                        /* number of processes */

int turn;                               /* whose turn is it? */
int interested[N];                      /* all values initially 0 (FALSE) */

void enter_region(int process);         /* process is 0 or 1 */
{
     int other;                         /* number of the other process */

     other = 1 – process;               /* the opposite of process */
     interested[process] = TRUE;        /* show that you are interested */
     turn = process;                    /* set flag */
     while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)          /* process: who is leaving */
{
     interested[process] = FALSE;       /* indicate departure from critical region */
}
```

Recap

- Solutions of Mutual Exclusion
  - Hardware Support
    - special instruction: TSL (Test and Set Lock)

```
enter_region:
        TSL REGISTER,LOCK          | copy lock to register and set lock to 1
        CMP REGISTER,#0            | was lock zero?
        JNE enter_region           | if it was not zero, lock was set, so loop
        RET                        | return to caller; critical region entered

leave_region:
        MOVE LOCK,#0               | store a 0 in lock
        RET                        | return to caller
```

Recap

- Solutions of Mutual Exclusion
  - Hardware Support
    - special instruction: XCHG (exchange)

```
enter_region:
        MOVE REGISTER,#1           | put a 1 in the register
        XCHG REGISTER,LOCK         | swap the contents of the register and lock variable
        CMP REGISTER,#0            | was lock zero?
        JNE enter_region           | if it was non zero, lock was set, so loop
        RET                        | return to caller; critical region entered

leave_region:
        MOVE LOCK,#0               | store a 0 in lock
        RET                        | return to caller
```

Recap

- Pthread Locks
  - Interface

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

  - Lock Intializaiton

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
```

```
int rc = pthread_mutex_init(&lock, NULL);
assert(rc == 0); // always check success!
```

  - alternatives

```
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_timelock(pthread_mutex_t *mutex,
                           struct timespec *abs_timeout);
```

Agenda

- ~~Recap~~
- Inter-Process Communication III
  - Semaphore
  - Conditional Variables

Semaphore

- Synchronization method that provides more sophisticated ways (than mutex locks) for process to synchronize their activities
- Semaphore **S** – integer variable
  - Can only be accessed via two indivisible (atomic) operations
    - **down()** and **up()**
      - originally called **P()** and **V()**
      - also called **wait()** and **signal()**

Semaphore

- Semaphore **S**
  - Definition of the **down()** operation

```
down(S) {
    while (S <= 0)
       ; // busy waiting
    S--;
}
```

  - Definition of the **up()** operation

```
up(S) {
    S++;
}
```

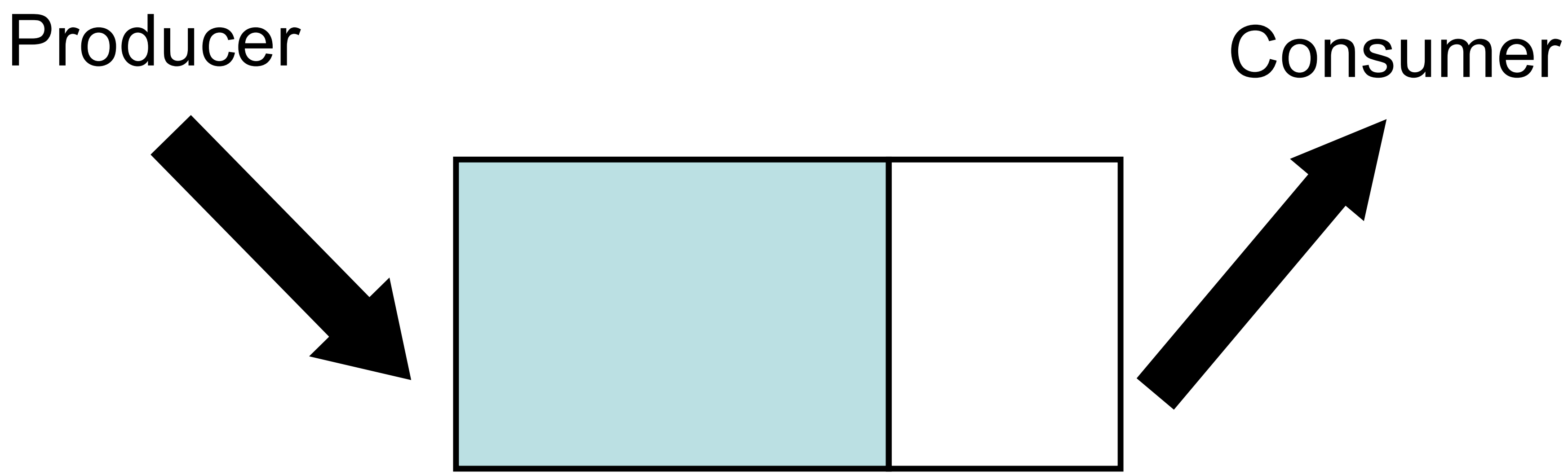
Semaphore

- **Counting semaphore**
  - integer value can range over an unrestricted domain
- **Binary semaphore**
  - integer value can range only between 0 and 1
  - same as a **mutex lock**
- Can solve various synchronization problems
  - e.g., consider P1 and P2 that require S1 to happen before S2
    - Create a semaphore “synch” initialized to 0

```
P1:
  S1; up(synch);
P2:
  down(synch); S2;
```

Producer-Consumer Problem

- Shared bounded buffer
  - A “Producer” process inserts item
  - A “Consumer” process removes item



Producer-Consumer Problem

- Solution with semaphore

```
#define N 100                    /* number of slots in the buffer */
typedef int semaphore;           /* semaphores are a special kind of int */
semaphore mutex = 1;             /* controls access to critical region */
semaphore empty = N;             /* counts empty buffer slots */
semaphore full = 0;              /* counts full buffer slots */
```

Producer-Consumer Problem

- Solution with semaphore

```
void producer(void)
{
        int item;

        while (TRUE) {                          /* TRUE is the constant 1 */
                item = produce_item( );         /* generate something to put in buffer */
                down(&empty);                   /* decrement empty count */
                down(&mutex);                   /* enter critical region */
                insert_item(item);              /* put new item in buffer */
                up(&mutex);                     /* leave critical region */
                up(&full);                      /* increment count of full slots */
        }
}
```

Producer-Consumer Problem

- Solution with semaphore

```
void consumer(void)
{
        int item;

        while (TRUE) {                        /* infinite loop */
                down(&full);                  /* decrement full count */
                down(&mutex);                 /* enter critical region */
                item = remove_item( );        /* take item from buffer */
                up(&mutex);                   /* leave critical region */
                up(&empty);                   /* increment count of empty slots */
                consume_item(item);           /* do something with the item */
        }
}
```

Use Semaphore with Caution

- **Deadlock**
  - two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes
  - Eg., Let ***S*** and ***Q*** be two semaphores initialized to 1

P_0	P_1
down(S);	down(Q);
down(Q);	down(S);
...	...
up(S);	up(Q);
up(Q);	up(S);

Use Semaphore with Caution

- **Starvation** – indefinite blocking
  - A process may never be removed from the semaphore queue in which it is suspended

- **Priority Inversion** – Scheduling problem when lower-priority process holds a lock needed by higher-priority process

Conditional Variables

- Allows a thread to wait till a condition is satisfied
- Testing the condition must be done within a mutex
- A mutex is associated with every condition variable

Thread call	Description
Pthread_cond_init	Create a condition variable
Pthread_cond_destroy	Destroy a condition variable
Pthread_cond_wait	Block waiting for a signal
Pthread_cond_signal	Signal another thread and wake it up
Pthread_cond_broadcast	Signal multiple threads and wake all of them

Conditional Variables

- Comparison with semaphore
  - If a signal is sent to a conditional variable on which no thread is waiting, the signal is lost
  - Semaphore will accumulate ‘signals’ by up()

- Comparison with mutex lock
  - Mutex lock is good to guarantee mutual exclusion
    - Allow and block access to the critical regions
  - Conditional variables block threads due to some condition not met

Conditional Variables

- Typical Usage

pthread_cond_t **condition_variable;**
pthread_mutex_t **mutex;**

Waiting Thread:

```
pthread_mutex_lock(&mutex);

while (condition not satisfied) {
    pthread_cond_wait(
      &condition_variable,
      &mutex);
  }

pthread_mutex_unlock(&mutex);
```

Signaling Thread:

```
pthread_mutex_lock(&mutex);

/* change variable value */

if (condition satisfied) {
     pthread_cond_signal(
        &condition_variable);
}

pthread_mutex_unlock(&mutex);
```

Conditional Variables

- Example
  - Write a program using two threads
    - Thread 1 prints “hello”
    - Thread 2 prints “world”
    - Thread 2 should wait till thread 1 finishes before printing
  - Use condition variables

Conditional Variables

- Example

```
int    thread1_done = 0;

pthread_cond_t    cv;
pthread_mutex_t  mutex;
```

Thread 1:

```
printf("hello ");

pthread_mutex_lock(&mutex);
thread1_done = 1;
pthread_cond_signal(&cv);
pthread_mutex_unlock(&mutex);
```

Thread 2:

```
pthread_mutex_lock(&mutex);

pthread_cond_wait(&cv, &mutex);

printf(" world\n");
pthread_mutex_unlock(&mutex);
```

Conditional Variables

- Example

```
int    thread1_done = 0;

pthread_cond_t    cv;
pthread_mutex_t  mutex;
```

Thread 1:

```
printf("hello ");

pthread_mutex_lock(&mutex);
thread1_done = 1;
pthread_cond_signal(&cv);
pthread_mutex_unlock(&mutex);
```

Thread 2:

```
pthread_mutex_lock(&mutex);

while (thread1_done == 0) {
    pthread_cond_wait(&cv,
    &mutex);
}

printf(" world\n");
pthread_mutex_unlock(&mutex);
```

Agenda

- ~~Recap~~
- ~~Inter-Process Communication III~~
  - ~~Semaphore~~
  - ~~Conditional Variables~~

Questions?

*acknowledgement: slides include content from “Modern Operating Systems” by A. Tanenbaum, “Operating Systems Concepts” by A. Silberschatz etc., “Operating Systems: Three Easy Pieces” by R. Arpaci-Dusseau etc., and anonymous pictures from internet.