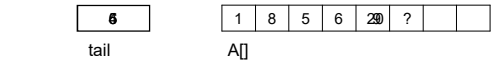
CSC 452: InterProcess Communication

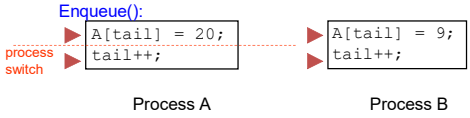
Dr. Jonathan Misurda
jmisurda@cs.arizona.edu
http://www.u.arizona.edu/~jmisurda

Race Condition

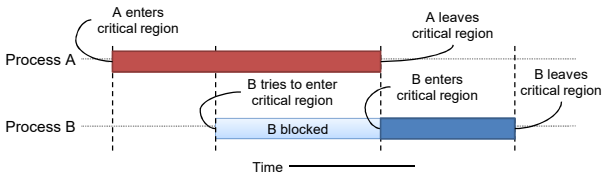
Shared Data:



Enqueue():

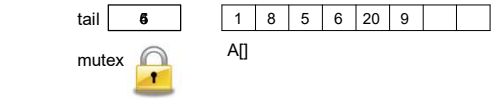


Critical Regions



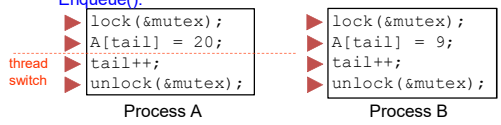
Critical Sections

Shared Data:



Enqueue():

Blocked!



pthread_mutex_t

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

int tail = 0;
int A[20];

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void enqueue(int value)
{
        pthread_mutex_lock(&mutex);
        A[tail] = value;
        tail++;
        pthread_mutex_unlock(&mutex);
}
```

Goals

- No two processes can be inside their critical regions at the same time
- No assumptions about CPU speed or number of CPUs
- No process outside its critical region may block another process
- No process should have to wait forever to enter its critical region

Strict Alternation

Process A

```
while (TRUE) {
  while (turn != 0)
    ; /* loop */
  critical_region ();
  turn = 1;
  noncritical_region ();
}
```

Process B

```
while (TRUE) {
  while (turn != 1)
    ; /* loop */
  critical_region ();
  turn = 0;
  noncritical_region ();
}
```

Hardware Support

```
int lock = 0;
```

Code for process P_i

```
while (1) {
  while (TestAndSet(lock))
    ;
  // critical section
  lock = 0;
  // remainder of code
}
```

Code for process P_i

```
while (1) {
  while (Swap(lock,1) == 1)
    ;
  // critical section
  lock = 0;
  // remainder of code
}
```

Busy Waiting

```
#define FALSE   0
#define TRUE    1
#define N       2       // # of processes
int interested[N];      // Set to 1 if process j is interested
int last_request;       // Who requested entry last?

void enter_region(int process)
{
    int other = 1-process;            // # of the other process
    interested[process] = TRUE;       // show interest
    last_request = process;           // Set it to my turn
    while (interested[other]==TRUE && last_request == process )
      ; // Wait while the other process runs
}

void leave_region (int process)
{
    interested[process] = FALSE;      // I'm no longer interested
}
```

Producer/Consumer Problem

Shared variables

```
#define N 10;

int buffer[N];
int in = 0, out = 0, counter = 0;
```

Producer

```
while (1) {
  if (counter == N)
    sleep();

  buffer[in] = ... ;
  in = (in+1) % N;

  counter++;

  if (counter==1)
    wakeup(consumer);
}
```

Consumer

```
while (1) {
  if (counter == 0)
    sleep();

  ... = buffer[out];
  out = (out+1) % N;

  counter--;

  if (counter == N-1)
    wakeup(producer);
}
```

Condition Variables

- A condition under which a thread executes or is blocked
- pthread_cond_t
- pthread_cond_wait (condition, mutex)
- pthread_cond_signal (condition)

Producer/Consumer

```
#define N 10
int buffer[N];
int counter = 0, in = 0, out = 0, total = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t prod_cond = PTHREAD_COND_INITIALIZER;
pthread_cond_t cons_cond = PTHREAD_COND_INITIALIZER;
```

```
void *producer(void *junk) {
   while(1) {
      pthread_mutex_lock(&mutex);
      if( counter == N )
         pthread_cond_wait(&prod_cond,
                           &mutex);

      buffer[in] = total++;
      printf("Produced: %d\n",
                        buffer[in]);
      in = (in + 1) % N;
      counter++;

      if( counter == 1 )
         pthread_cond_signal(&cons_cond);

      pthread_mutex_unlock(&mutex);
   }
}
```

```
void *consumer(void *junk) {
   while(1) {
      pthread_mutex_lock(&mutex);

   if( counter == 0 )
      pthread_cond_wait(&cons_cond,
                        &mutex);

      printf("Consumed: %d\n",
                        buffer[out]);
      out = (out + 1) % N;
      counter--;

      if( counter == (N-1) )
         pthread_cond_signal(&prod_cond);

      pthread_mutex_unlock(&mutex);
   }
}
```