CSC552 – Advanced UNIX Programming

Critical Sections

Dr. L. Frye
Kutztown University

Critical Section

- Critical section
- Mutual exclusion

- Cooperation of programs and users
- How can this be handled?

- Critical section problem

Examples

- critical/simplechain.c
- critical/chaincritical.c
- Why is the highlighted section a critical section?
  - What is the problem when run with various values for delay argument?
  - How can it be corrected?

Programming Sections

- Entry section
- Critical section
- Exit section

- Don’t postpone indefinitely thread trying to enter critical section
- Threads should make progress

IPC Revisited

- POSIX/XSI
  - Message queues
  - Semaphore sets
  - Shared memory
- IPC objects
  - get functions
- sys/msg.h
- sys/sem.h
- sys/shm.h

IPC Object Key

- Let system pick key
- Pick a key directly
- Ask system to generate a key – ftok()

IPC Shell Commands

- ipcs
- ipcrm

- Race condition

Example – Print Spooler

- Spooler Directory
  - Large number of slots
    - Filename to be printed
- Printer Daemon
- Two shared variables
  - out
  - in

- How can this be avoided?

Possible Solutions

- Disable interrupts
- Lock variables
- Strict alternation
- Peterson's solution

Possible Solutions

- Disable Interrupts
  - Simple
  - Users should not be able to turn off interrupts
  - Only affects one CPU
- Lock Variables
  - Single, shared variable
  - Race conditions

Strict Alternation

- Process A

```
while (true)  {
    while (turn != 0) ;
    critical_region();
    turn = 1;
    noncritical_region();
}
```

Process B

```
while (true)  {
    while (turn != 1) ;
    critical_region();
    turn = 0;
    noncritical_region();
}
```

- What is the problem here?

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 FALSE */
void enter_region(int process)
{
   int other;     /* number of the other process */
   other = 1 - process; /* the opposite of process */
   interested[process] = TRUE;          /* show interest */
   turn = process;                              /* set flag */
   while (turn == process && interested[other] == TRUE) ;
}
void leave_region(int process)          /* process who is leaving */
{
    interested[process] = FALSE;        /* leaving critical section */
}
```

Producer-Consumer Problem

- Producer – adds information
- Consumer – removes information

- Print server example

- Where does the problem arise in the producer-consumer problem?

Producer Code

```
void producer()
{
   while (TRUE)  {              /* loop forever */
      produce_item();           /* generate next item */
      if (count == N)  sleep();    /* buffer full? */
      enter_item();             /* put item in buffer */
      count = count + 1;
      if (count == 1) wakeup(consumer);   /* empty? */
   }
}   /* end producer */
```

Consumer Code

```
void consumer()
{
   while (TRUE)  {              /* loop forever */
      if (count == 0) sleep (); /* buffer empty? */
      remove_item();            /* take item out of buffer */
      count = count - 1;
      if (count == N-1) wakeup(producer);     /* full? */
      consume_item();           /* print item */
   }
}   /* end consumer */
```

Question

- When can the race condition occur?