

Inter-Process Communication

OS 2003 1

Issues

- How a process can pass information to another
- Make sure processes don't get into each others' way
- Sequencing and dependencies

OS 2003 2

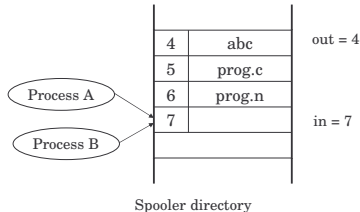
The issues...

- They apply to threads as well
- Communication: easy for threads (common address space)
- Remaining two issues apply to thread as to processes

OS 2003 3

Race condition

- Example: printer spooler (a daemon)



OS 2003 4

Race condition

- Two processes reading/writing on the same data and the result depends on who runs precisely when is called a *race condition*
- Since obviously we'd like computation to be deterministic

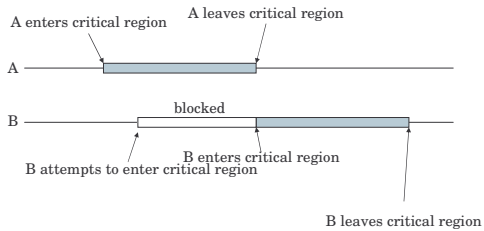
OS 2003 5

Critical regions

- *Mutual exclusion*
- The part of the program where the shared memory (or something else) is accessed is called a *critical section*
- This is not enough (more rules):
 - Not two processes simultaneously in their critical regions
 - No assumptions may be made about speed and number of CPUs
 - No process running outside its critical region may block another process
 - No process should have to wait forever to enter its critical region

OS 2003 6

Ideally



OS 2003

7

Many solutions...

- Disabling interrupts
- Locks
- TSL instruction (hardware)
- Semaphores
- Mutexes
- Monitors
- Message passing
- ...

OS 2003

8

Disabling interrupts

- Simplest solution
- CPU switches from process to process only when an interrupt occurs (e.g. the clock interrupt)
- This approach can be taken by the kernel
- Should the OS trust the user in disabling/enabling interrupts? Too dangerous!

OS 2003

9

Locks

- A lock variable (alone it doesn't work)
- Strict alternation (no two in the critical region, not convenient)

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

OS 2003

10

Peterson's solution

```
#define FALSE 0
#define TRUE 1
#define N 2

int turn;
int interested[N]; // initialized = 0

void enter_region(int process)
{
    int other;
    other = 1 - process;
    interested[process] = TRUE;
    turn = process;
    while (turn == process && interested[other] == TRUE) ;
}

void leave_region(int process)
{
    interested[process] = FALSE;
}
```

OS 2003

11

TSL instruction

- TSL RX, LOCK (test and set lock)
- Reads the content of LOCK into RX and stores a non-zero value into LOCK atomically (can't be interrupted)

OS 2003

12

Example

```
enter_region:
    TSL REGISTER, LOCK
    CMP REGISTER, #0
    JNE enter_region
    RET

leave_region:
    MOVE LOCK, #0
    RET
```

OS 2003

13

Semaphores

- An **atomically** accessible counter. Similar to a lock but with multiple values and possibly blocking a process without busy-waiting
- There are two operations possible:
 - Up, Down
- Down, if 0 the process will go to sleep otherwise it decrements the semaphore and continues execution
- Up, increments the semaphore, if a process is sleeping on the semaphore, it is awakened, the caller never blocks

OS 2003

14

Example consumer-producer

```
#define N 100
typedef int semaphore; // with a bit of\
    imagination

semaphore mutex = 1;
semaphore empty = N;
semaphore full = 0;

void producer(void)
{
    int item;
    while (TRUE)
    {
        item = produce_item();
        down(&empty);
        down(&mutex);
        insert_item(item);
        up(&mutex);
        up(&full);
    }
}

void consumer(void)
{
    int item;
    while (TRUE)
    {
        down(&full);
        down(&mutex);
        item = remove_item();
        up(&mutex);
        consume_item(item);
    }
}
```

OS 2003

15

Mutexes

- Semaphores with binary values
- What's nice? Simpler implementation than semaphores
- Of course, a semaphore can be made to behave as a mutex and vice-versa a mutex is enough to implement a semaphore

OS 2003

16

Monitors

- Abstract construct (a package):
 - It's a sort of class (in fact there's something similar in Java)
 - Monitor's data is *private*
 - Only one process can be **active** in a monitor at a given time
 - Condition variables: **wait** and **signal** primitives (equivalent to **down** and **up**)

OS 2003

17

Part of an example...

```
monitor ProducerConsumer
    condition full, empty;
    integer count;

    // PROCEDURES_HERE ()
    // it's guaranteed that no process can change
    // count at the same time, just need to check
    // the full and empty conditions

    count = 0;
end monitor;
```

OS 2003

18

Message passing

- Why? Distributed systems for example
 - send(destination, &message)
 - receive(source, &message)

OS 2003

19

Issues with message passing

- Acknowledgement (message)
 - We need to be sure a message is not lost otherwise synchronization will go berserker
 - Message numbering
 - A good part of the study on computer networks
- Authentication
 - Make sure only who's supposed to receive the message actually receives it and vice-versa

OS 2003

20

Example of message passing

```
#define N 100

void producer(void)
{
    int item;
    message m;

    while (TRUE)
    {
        item = produce_item();
        receive(consumer, &m);
        build_message(&m, item);
        send(consumer, &m);
    }
}

void consumer(void)
{
    int item, i;
    message m;

    for (i=0; i<N; i++) send(producer, &m);
    sends N EMPTYIES

    while (TRUE)
    {
        receive(producer, &m);
        item = extract_item(&m);
        send(producer, &m);
        consume_item(item);
    }
}
```

OS 2003

21

Access to database

- Many readers
- Only one writer
- Issues: no write until all readers are out, but try not to accept other readers if a write is pending!

OS 2003

22