Intro to Computer Systems

CPU

RAM

Bus

1/0 devices

DMA Transfer

Memory Access

Arithmetic/Logic Unit

CPU

Program Counter

Instructor Register

Control Unit

PC

MAR

MDR

IR

CU

ALU

Start

Instruction Execution Cycle.

Fetch instruction

Decode instruction

Execute inst.

Store result in RAM if needed
Intro to Computer Science
Interrupts

A mechanism by which the normal processing of the CPU is changed.

- Identify interrupt
- Identify interrupt service program
- Resume normal execution

Program 1

\[\text{interrupt}\]

Can disable interrupts.

i/o accesses

Data transfer between i/o device & RAM or CPU.

- Programmed i/o - busy waiting
- Interrupt driven
- Direct Memory Access
Process Management

Process? - Program in Execution

Needs: CPU, Memory, Files, I/O...

OS Responsibilities

(i) Creation / Termination
(ii) Suspension / Resumption
(iii) Process Synchronization
(iv) Process Scheduling
(v) IPC - Communication
(vi) Deadlock Handling

one Process e - Process Control Block [PCB]

Contents: (a) Memory info
(b) Scheduling info
(c) PC int in Unix
(d) State
(e) id [Process id, User id, ...]
(f) Accounting info
(g) The status info
Process Management

Proces 5 - Program is created

Need: CPU, Memory, Files, I/O

02 Resources

[Creation of Process]
[Supervision / Termination]
[Process, Program, and Data]
[Process Control]
[Process Communication]
[Process Synchronization]
[Process Management]

Note: Control Block I Bec...

Criteria:
(a) Memory initial
(b) Scheduler
(c) "next"
(d) State
(e) Branch
(f) Program
(g) Next state
(h) Name

Process switch

P

\}

Save P1's state
in P1's PCB

Load P2's state

Process State Diagram

ready

New

Scheduler picks

interrupted

Waiting for an event

Event happens

Blocked

Terminated
Front Switch

R

L

Some 6'7' Poster

1 6'6' Poster

Front Poster Diagram

Original 6'6" Poster

Subject

Knowledge

Interpretation

Goose

Feedback

Level 1

Feedback

Level 1
Process Creation

Process Creation in Unix

fork() system call

Creates an identical copy
(Parent) old process creates
(child) new process

fork()

create new PCB
Copy fields of PCB
Copy RAM

return

9/8/13

main()

fork();

i++;

fork();
Process Creation

Create a new process

fork() system call

Create a new process

(continued) Old parent process

Create from parent

Child Program
0) Differentiating parent from child?

1) child is a copy of parent
   - overhead?
   Not a lot of copying.

---

**Processes**

- **Text**
  - Executable
    - Code: not copied.

- **Data**
  - Variables
    - Cannot be shared by parent & child

- **Stack**
  - Activation Record

---

Lazy copying

<table>
<thead>
<tr>
<th>partition Data region into equal-sized slices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Each slice has one bit called Copy-on-write</td>
</tr>
</tbody>
</table>
fork() returns pid of child to parent & to child.

main() {
    int pid;
    if ((pid = fork()) == 0) {
        exec(); // child code
    } else {
        // parent code
        printf("%d", pid);
    }
}

9/15/13 Ass
Equivalent Assembly Code.

```assembly
CALLP
fork:

MOV RY, PID; RV contains ret
; value of function

; parent gets pid of child as ret value
; child gets 0 as ret value.
CMP PID, #0
BNE PAR

[ child code

BR Common

PAR: ; parent code

; end of parent code

Common
```

only one process


Concurency

Producer - Consumer problem

Bounded Buffer Problem

Buffer

```
item_type Buffer[0 \&lt; \&lt; k];
int in, out = 0;
```

Producer

```
while (in \lt \&lt; k)
    produce an item in item;
    while (in++ \mod k == out) wait;
    Buffer[in] = item;
    in++ \mod k
```

Consumer

```
while (in == out) wait;
    item = Buffer[out];
    out = out++ \mod k;
    consume item;

    /* in == out => buffer is empty */
```
Buffer

Goto buffer

Producer - Consumer Problem

Problems

Producer

Consumer

buffer

\[ \text{Buffer Capacity: } C \]

\[ \text{Producer: } \text{in} + 1, \text{out} = 0 \]

\[ \text{Consumer: } \text{in} = \text{out} \]

\[ \text{Buffer State: } B \]

\[ \text{Producer: } B = \text{out} \]

\[ \text{Consumer: } B = \text{in} \]

\[ \text{Buffer Empty: } B = 0 \]

\[ \text{Buffer Full: } B = C \]

\[ \text{Producer: } \text{in} = \text{out} + 1 \]

\[ \text{Consumer: } \text{in} = \text{out} \]

\[ \text{Buffer Empty: } B = 0 \]

\[ \text{Buffer Full: } B = C \]

\[ \text{Producer: } \text{in} = \text{out} + 1 \]

\[ \text{Consumer: } \text{in} = \text{out} \]

\[ \text{Buffer Empty: } B = 0 \]

\[ \text{Buffer Full: } B = C \]

\[ \text{Producer: } \text{in} = \text{out} + 1 \]

\[ \text{Consumer: } \text{in} = \text{out} \]

\[ \text{Buffer Empty: } B = 0 \]

\[ \text{Buffer Full: } B = C \]
\((\text{in} + 1 \mod k) = \text{out} : \text{we pretend that the buffer is full. [Buffer has one empty space]}

Why not use a shared variable \text{Count} which counts number of items in buffer?

\text{Consumer} \hspace{2cm} \text{Producer}

\text{Count} \downarrow \hspace{2cm} \text{Count} \uparrow

Initially \hspace{1cm} \text{Count} 5

1. MOV COUNT, R1
2. MOV COUNT, R5
5. DEC R1
3. INC R5
6. MOV R1, COUNT
4. MOV R5, COUNT
Prepare that the butter is full. I butter my one piece empty slice.

Which nut use a 2-point

Numeral of count which counts

Initialize count a

- Count
- Count
- Count
- Count
- Count
- Count

When count 61
2 decade 6
3 decade 6
5 when count 61
CRITICAL SECTIONS

inside critical sections, processes may use shared variables.

Conditions

1. MUTEX: When one process is executing in its critical section, no other process may execute in its critical section.

2. No indefinite wait

3. No unnecessary wait

Attempts: only 2 processes

1. Use turn variable \( P_i + P_j \) (initialized to \( i \) or \( j \))

\[ P_i \]

while (turn != i) wait;

\[ C_S_i \]

[critical section of \( P_i \)]

turn = j;
Critical Sections

Include Critical Sections Process

May not share variables

Continuing a section is its critical section

Critically section

No interference with

No process necessarily

set to only 1 process

Attempts to

Use to monitor

(i.e. at initialization)

N

Critical section

if (m != i) then

Write (m = i)
boolean

#2 flag_i, flag_j = 0;

// flag_i = 1 => Pi is inside CS_i

while (flag_j) wait;

flag_i = true;

CS_i

flag_i = false;
flag_i = false

while (flag_i) wait;

flag_j = true;

CS_j

flag_j = false;
# Problem

1. Let \( L = 1 \). If \( n \) is odd, then \( L = 0 \).

\[
\frac{L}{n} + \frac{L}{n} = \frac{2L}{n}
\]

\[
\frac{2L}{n} = \frac{2L}{n} - \frac{2L}{n} = \frac{2L}{n}
\]

\[
\frac{2L}{n} = \frac{2L}{n}
\]

\[
\frac{2L}{n} = \frac{2L}{n}
\]

\[
\frac{2L}{n} = \frac{2L}{n}
\]