CS341: Operating System

Process Concepts

Lect10 : 21th Aug 2014

Dr. A. Sahu
Dept of Comp. Sc. & Engg.
Indian Institute of Technology Guwahati

1

Outline

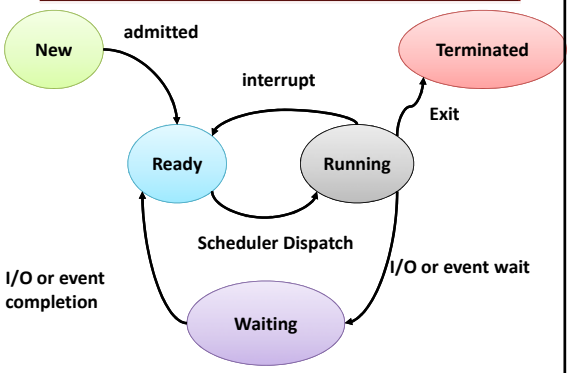
- Process Concepts
  - Process States, PCB
  - System call related to Process & C examples
  - **IPC & Thread in Future class**
- Scheduling: **Theoretical Analysis**

2

Recap

3

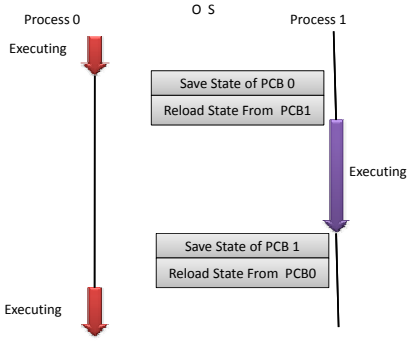
Process State: State Diagram



PCB: Info. related to Process

- Process state
- Program counter, CPU registers
- CPU scheduling info.- priorities, scheduling queue pointers
- Memory-management info: memory allocated to the process
- Accounting info. – CPU used, clock time elapsed since start, time limits
- I/O status info. – I/O devices allocated to process, list of open files

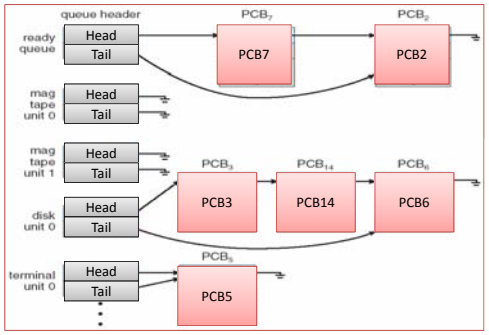
CPU Switch From Process to Process



Process Scheduling

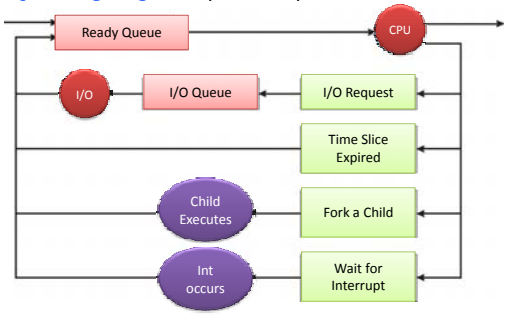
- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the various queues

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling

- **Queueing diagram** represents queues, resources, flows



Schedulers

- **Short-term scheduler** (or **CPU scheduler**)
  - Selects which process should be executed next and allocates CPU
  - Short-term scheduler is invoked frequently (milliseconds) ⇒ (must be fast)
- **Long-term scheduler** (or **job scheduler**)
  - Selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) ⇒ (may be slow)
  - The long-term scheduler controls the **degree of multiprogramming**

Schedulers

- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good ***process mix***

Context Switch

- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB ➔ the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU ➔ multiple contexts loaded at once

Many time Context switch code written Manually in ALP. Compiler Generated code may not be efficient

Next…

13

Operations on Processes

- One should know how to create/delete process
- System must provide mechanisms for:
  - process creation
  - process termination

Process Creation

- **Parent** process create **children** processes
  - which, in turn create other processes, forming a **tree** of processes
- Generally, process identified and managed via a **process identifier** (**pid**)

ps : report a snapshot of current process

```
[asahu@asahu ~]$ ps -A
  PID TTY          TIME CMD
    1 ?        00:00:02 systemd
    2 ?        00:00:01 kthreadd
    3 ?        00:00:22 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    7 ?        00:00:00 kworker/u:0H
    8 ?        00:00:11 migration/0
    9 ?        00:00:07 watchdog/0
   10 ?        00:00:07 migration/1
   12 ?        00:00:00 kworker/1:0H
   13 ?        00:00:20 ksoftirqd/1
   14 ?        00:00:17 watchdog/1
   15 ?        00:00:12 migration/2
   17 ?        00:00:00 kworker/2:0H
   18 ?        00:00:22 ksoftirqd/2
   19 ?        00:00:12 watchdog/2
```

16

pstree

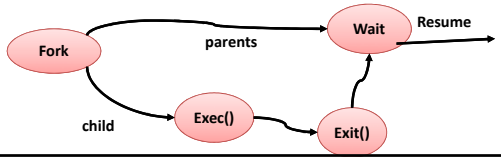
```
[asahu@asahu ~]$ pstree
systemd─┬─NetworkManager───{NetworkManager}
        ├─VBoxSVC─┬─VirtualBox───18*[{Virtual
        │         └─11*[{VBoxSVC}]
        ├─VBoxXPCOMIPCD
        ├─abrt-dump-oops
        ├─abrtd
        ├─accounts-daemon───{accounts-daemo}
        ├─acpid
        ├─at-spi-bus-laun───2*[{at-spi-bus-la
        ├─atd
        ├─auditd─┬─audispd─┬─sedispatch
        │        │         └─{audispd}
        │        └─{auditd}
        ├─avahi-daemon───avahi-daemon
        ├─colord───{colord}
        ├─console-kit-dae───63*[{console-kit-
```

Process Creation

- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **`fork()`** system call creates new process
  - **`exec()`** system call used after a **`fork()`** to replace the process' memory space with a new program



C Program Forking Separate Process

```
int main (){
        pid_t pid;
        pid=fork();
        If(pid<0){
                printf("fork failed");
                return 1;
        }
        else if (pid==0){
                execlp("/bin/ls","ls",NULL);
        }
        else {
                wait(NULL);
                printf("Child complete")I
        }
        return 0;
}
```

How to terminate a Process

- ☺ ☺ ☺ ☺
  - **Click the Cross symbol located on right side of the application window**
- In linux command mode
  - **Send termination signal to process**

$kill <pid>

21

Process Termination

- Process executes last statement and then asks the OS to delete it using the **`exit()`** system call.
  - Returns status data from child to parent (via **`wait()`**)
  - Process' resources are deallocated by operating system

Process Termination

- Parent may terminate the execution of children processes using the **`abort()`** system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

CPU Scheduling

24

Scheduling Evaluation Metrics

- CPU utilization
  - Percentage of time the CPU is not idle
- Throughput
  - Completed processes per time unit
- Turnaround time
  - Submission to completion

Scheduling Evaluation Metrics

- Waiting time
  - time spent on the ready queue
- Response time
  - response latency
  - "response time" most important for interactive jobs (I/O bound)
- Predictability
  - variance in any of these measures

The right evaluation metric depends on the context

"The perfect CPU scheduler"

- Minimize latency
  - Response or job completion time
- Maximize throughput
  - Maximize jobs / time.
- Maximize utilization: keep I/O devices busy.
  - Recurring theme with OS scheduling
- Fairness: everyone makes progress, no one starves

Scheduling Algorithm Optimization Criteria

- Maximize
  - CPU utilization
  - Throughput
- Minimize
  - Turnaround time
  - Waiting time
  - Response time

Problem: Cook of Restaurant

- You work as a short-order cook
  - Customers come in and specify which dish they want
  - Each dish takes a different amount of time to prepare
- Your goal:
  - minimize average time the customers wait for their food
- What strategy would you use ?

Assumption

- Task/Process/Job used interchangbly
- Let all tasks in memory
- Let all tasks arrived at time 0
- Let all tasks don't do any IPC
- All task are independent
- All tasks don't require any other resource other then CPU

Scheduling Problem: Example

- Let 10 independent tasks
  - Execution time i_{th} task is t_i
- 1 CPU
- Criteria : Minimize Cmax
  - Cmax= finishing time of last finished task
- Sol : Execute in any order

32

Scheduling Problem: Example

- Let 10 independent tasks
  - Execution time i_{th} task is t_i
  - Also ti is divided in to two part
    - D_i (I/O of Device) and C_i (CPU time)
- 1 CPU + 1 Device
  - A process is allowed to both D or C simultaneously
  - A process is not allowed both D or C simultaneously
  - Order of Di an Ci of Ti is independent
- Criteria : Minimize Cmax
- Sol : Based on Policy

33