Operating Systems

Lecture #6: Process Management

Written by David Goodwin
based on the lecture series of Dr. Dayou Li
and the book *Understanding Operating Systems* 4^{th} *ed.*
by *I.M.Flynn and A.McIver McHoes* (2006)

Department of Computer Science and Technology,
University of Bedfordshire.

Operating Systems, 2013

04^{th} March 2013

Outline

3

Introduction

PROCESS MANAGEMENT CONCEPTS

- Concept of a process
  - Terminology
    - **Job** (also known as program) is an inactive unit such as a file in disk
    - This entity contains at least two types of elements: program code and a set of data
    - **Process** (also called task) is an executed program code on the processor, that is, a part of an active entity
    - a **Thread** is a portion of a process that can be run independently
  - In multiprogramming environment, the processor is used by multiple programs/processes
  - Process manager needs to arrange the execution of the programs/processes (to make a schedule for them) to promote the most efficient use of resources (including hardware and software) and to avoid competition and deadlock

Scheduling

Job and Process Scheduling

- Schedulers
  - Job scheduler
    - Initialise each job
    - Only concerned with selecting jobs from a queue of incoming jobs
    - Places them in a process queue (READY queue)
  - Process scheduler
    - Determines which jobs get the CPU, when, and for how long.
    - Also decides when processing should be interrupted
    - decides when each step will be executed
    - recognises when a job has concluded and should be terminated
  - Hierarchy
    - Job scheduler is the high-level scheduler
    - Process scheduler is the low-level scheduler

Process Sheduler

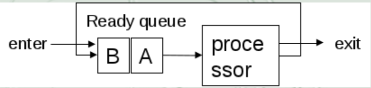
- Common trait among most computer programs: they alternate between CPU cycles and IO cycles
- Process scheduler is a low-level scheduler that assigns CPU the execute the processes of those jobs placed in the ready queue by the job scheduler.
- Although CPU cycles vary from program to program, there are some general types of jobs:
  - **IO-Bound** jobs e.g. printing a series of documents. Many brief CPU cycles and long IO cycles.
  - **CPU-Bound** jobs e.g. finding the first 300 prime numbers. Long CPU cycles and shorter IO cycles.
  - Total statistical effect approximates a Poisson distribution.
- Highly interactive environment has a third level - begin a middle-level scheduler - finding it advantagious to remove active jobs from memory; allowing jobs to complete faster.

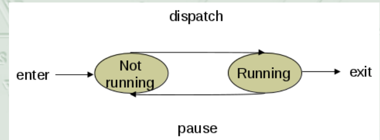
Process status

PROCESS STATUS

Two-state model

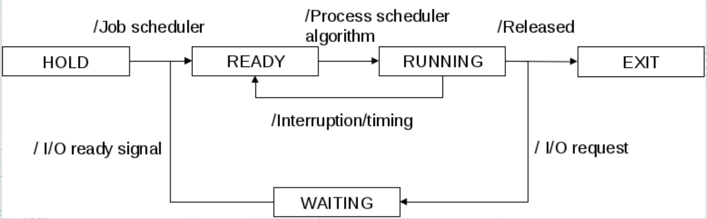
- Process queue
  - Processes are divided into threads
  - The threads form a "ready queue" waiting for being processed in the processor
- Two-state model
  - A process is in "running" state if it is executed on the processor
  - It is in "not running" state otherwise





PROCESS STATUS

- Five-state model
  - States
    - **Running**: the process is currently executed on the processor
    - **Ready**: a process is prepared to execute
    - **Waiting**: a process waits for resources
    - **Hold**: a process is just created
    - **Exit**: a process is released from pool of executable programs by OS



PROCESS STATUS

- State transitions:
  1. HOLD to READY:
     - When OS is prepared to take an additional process. Initiated by job scheduler; availability of memory and devices checked.
  2. READY to RUNNING:
     - Process scheduler chooses a ready process to execute.
  3. RUNNING to EXIT:
     - Currently executed process has completed or a run time error. Initiated by job or process scheduler.
  4. RUNNING to WAIT:
     - Handled by process scheduler, initiated by job instruction for I/O or other resources request.
  5. WAIT to READY:
     - Handled by process scheduler, initiated when I/O or other resources become available.
  6. RUNNING to READY:
     - Handled by process scheduler; maximum allowable time is reached or other criterion e.g. a priority interrupt.

Process control block

Process Control Blocks

Process control block (PCB): Any process is uniquely characterised by a list of elements:

- **Identifier**: a unique identifier which distinguishes a process from others
- **State**: a process can have different states
- **Priority**: this shows the priority level of a process relative to other process
- **Program counter**: the address of the instruction to be executed next of a process (program code)
- **Memory pointers**: include those point to program code and data associated of a process, plus any memory blocks shared with other processes
- **Context data**: These are date that are presented in registers in the processor while a process is running
- **I/O status information**: includes I/O requests and the availability of files and I/O devices
- **Accounting information**: includes time and number countered

Identifier
State
Priority
Program counter
Memory pointers
Context data
I/O status info
Accounting info

Process State

- This contains all of the information needed to indicate the current state of the job such as:

process status word current instruction counter and register contents when a job isn't running.

register contents if the job has been interrupted and is waiting.

main memory information including the address where the job is stored.

resources information about all resources allocated to the job, being hardware units or files.

process priority used by systems with priority scheduling algorithm.

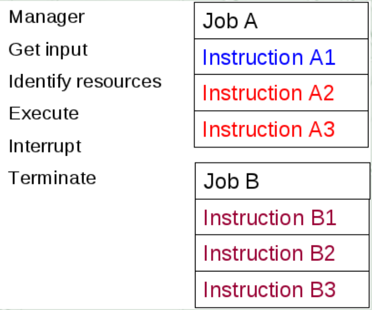
Accounting

- Information for billing purposes and performance measurement. Typical charges include:
  - Amount of CPU time used from beginning to end of execution.
  - Total time the job was in the system until exited.
  - Main storage occupancy - how long the job stayed in memory until it finished execution.
  - Secondary storage used during execution.
  - System programs used (compilers, editors, utilities, etc.)
  - Number and type of IO operations, including IO tranmission time (includes use of channels, control units, devices)
  - Time spent waiting for IO completion
  - Number of input records read, and number of output records written

Multithreading

- Multithreading
  - A process can be divided into multiple threads
    - Multithreading is a "condition" for a processor manager to make a schedule for multi-processes
  - An example of a process and threads

- Get input for Job A
- Identify resources
- Execute the process
- Interrupt
- Switch to Job B
- Get input for Job B
- Identify resources
- Execute the process
- Terminate Job B
- Switch back to Job A
- Resume executing interrupted process
- Terminate Job A



Process scheduling policies

Process scheduling policies

- A good process scheduling policy:
  - *Maximise throughput* - run as many jobs as possible in a given amount of time
  - *Minimise reponse time* - quickly turn around interactive requests
  - *Minimise turnaround time* - move entire jobs in and out of the system quickly
  - *Minimise waiting time* - move jobs out of the ready queue as quickly as possible
  - *Maximise CPU efficiency* - keep the CPU busy 100% of the time
  - *Ensure fairness for all jobs* - give every job an equal amount of CPU and IO time
- A scheduling strategy that interrupts the processing of a job and transfers the CPU to another job is **preemptive scheduling policy**
- The alternative is **nonpreemptive scheduling policy** which functions without external interrupts

Process scheduling algorithms

Process scheduling algorithms

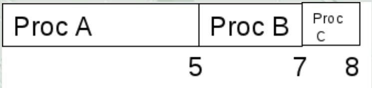
FCFS - First-come-first-served

- Batch environment
  - Jobs/processes arrive at the times that are too close to one another
  - Those jobs/processes are collected in batches
  - First-come-first-serve (FCFS)
    - A READY queue for storing ready processes that are initialised by Job scheduler
    - When a process is in RUNNING state, its execution will be completed in one go, that is, these is no waiting state in this algorithm
  - Average turnaround time (τ)
    - Total time needed for every process completed divided by the number of processes in the queue
    - Depends on how the processes are queued in the READY queue

Process scheduling algorithms

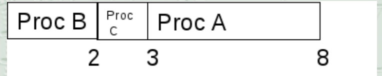
FCFS - First-come-first-served

- Example:
  - Process A has CPU cycle ($t_a = 5$ ms)
    Process B has CPU cycle ($t_b = 2$ ms)
    Process C has CPU cycle ($t_c = 1$ ms)
  - When the 3 processes become ready in the order of ABC



$$\tau = (5+7+8)/3 = 6.67$$

  - When the 3 processes become ready in the order of BCA



$$\tau = (2+3+8)/3 = 4.33$$

Process scheduling algorithms
SJF - Shortest job First

- Shortest job first [next] (SJF) [SJN]
  - Process that uses the shortest CPU cycle will be selected for running first
  - Example:

Processes:	A	B	C	D
CPU cycle (t_i):	5	2	6	4

Proc B	Proc D	Proc A	Proc C
2	6	11	17

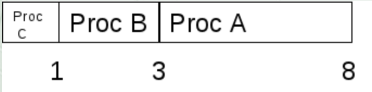
$$T_b = 2,\ T_d = 6,\ T_a = 11,\ T_c = 17$$
$$\tau = (T_b + T_d + T_a + T_c)/4 = 9$$

Process scheduling algorithms

SJF - Shortest job First

- Optimal in terms of occupying the least CPU time
  - Example:

Processes:	A	B	C
CPU cycle (t_i):	5	2	1



$$\tau = (1 + 3 + 8)/3 = 4$$

$$\tau = \frac{1}{n}\sum_{j=1}^{n} T_j = \frac{1}{n}\left(\sum_{j=1}^{i-1} T_j + \sum_{k=1}^{i-1} t_k + t_i + \sum_{k=1}^{i-1} t_k + t_i + t_{i+1} + \sum_{j=i+2}^{n} T_j\right)$$

$$\tau' = \frac{1}{n}\left(\sum_{j=1}^{i-1} T_j + \sum_{k=1}^{i-1} t_k + t_{i+1} + \sum_{k=1}^{i-1} t_k + t_i + t_{i+1} + \sum_{j=i+2}^{n} T_j\right)$$

$\square t_i > t_{i+1}$

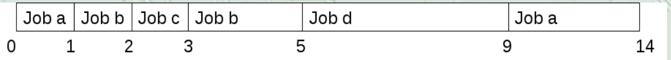
$\square \tau' < \tau$

Process scheduling algorithms

SRT - Shortest remaining time

- Real-time environment
  - Shortest remaining time (SRT)
    - Process that the mostly close to complete will be process first, and even this process can be pre-empted if a newer process in READY queue has a shorter complete time
    - Example

Arrival time:	0	1	2	3
Process:	a	b	c	d
CPU cycle:	6	3	1	4



$$\tau = (14 + 4 + 1 + 6)/4 = 6.25$$

Process scheduling algorithms
Round-Robin

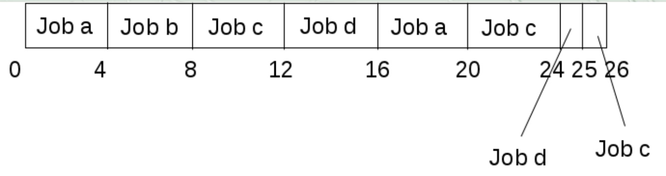
- Round robin
  - Time is divided into slices called **time quantum**
  - Processes in the READY queue will be processed in FIFO
  - Process scheduler will pick up a process from the front of the READY queue and allocate it to CPU when a time slide starts
  - When the time slide ends but the processing has not been complete, the scheduler will send the remaining part of the process back to the end of the READY queue
  - It can only be resumed when it gets to the front of the queue

Process scheduling algorithms

Round-Robin

- Example

Arrival time:	0	1	2	3
Process:	a	b	c	d
CPU cycle:	8	4	9	5



- $\tau = (20 + 7 + 24 + 22)/4 = 18.25$

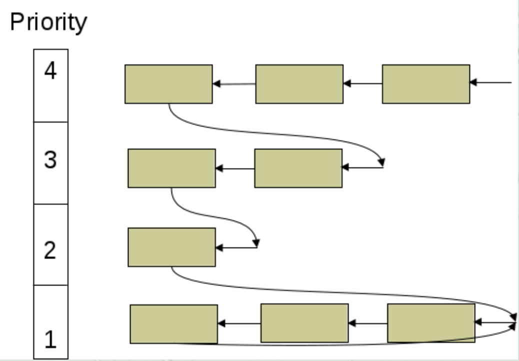
Process scheduling algorithms
Priority scheduling

- Priority scheduling
  - Allowing process with the highest priority to be processed first
  - The processing is not interrupted unless the CPU cycle of the process is completed or a natural wait occurs
  - If more than one process has the same priority, the one which joins the ready queue first is processed first
  - Priorities:
    - *Memory requirements* - jobs requiring large amounts of memory could be allocated lower priorities than those requiring small amounts of memory
    - *Number and type of device* - jobs requiring many devices would be allocated lower priorities
    - *Total CPU time* - jobs having long CPU cycle, or estimated run time, would be given lower priorities
    - *Amount of time already spent in the system* - some systems increase priority of jobs that have been in the system for an unusually long time. This is known as **aging**

Process scheduling algorithms

Multilevel feedback queue

- Multilevel feedback queue
  - When time out for a high-priority process, it is added to the tail of the queue consisting of lower priority level processes



Process scheduling algorithms

Multilevel feedback queue

- Priority inversion and inheritance
  - Priority inversion problem: a high-priority process waits on a low-priority process that is starved by a medium-priority process (for example, releasing some resources)
  - Example
    - Three processes, a, b and c with p(a)=2, p(b)=3 and p(c)=4
    - Process c waits on a
    - a is starved by b (b is never put in waiting state)
    - Process a cannot compete with b and therefore b is always running and a and c never get chance to run
    - It looks like c with p(c)=4 had lower priority than b with p(b)=3

Process scheduling algorithms

Multilevel feedback queue

- Solution – priority inheritance
  - Elevating the priority a to 4 so that it can compete with b and win in competition

process scheduling algorithms - comparison

- FCFS - nonpreemptive - batch - unpredictable turnaround - Easy to implement
- SJN - nonpreemptive - batch - indefinite postponement - minimise average waiting
- Priority scheduling - nonpreemptive - batch - indefinite postponement - fast completion of important jobs
- SRT - Preemptive - batch - overhead from context switching - fast completion of short jobs
- Round robin - Preemptive - interactive - requires selection of good time quantum - fair CPU allocation and reasonable response times for interactive users
- Multiple-level queues - Preemptive/nonpreemptive - batch/interactive - overhead from monitoring queues - flexible, counteracts indefinite postponement with aging, fair treatment of CPU-bound jobs by incrementing time quantums on lower priority queues

SUMMARY

Key Terms

aging
context switching
CPU-bound
first come, first served
high-level scheduler
IO-bound
indefinite postponement
interrupt
interrupt handler
job scheduler
job status
low-level scheduler
middle-level scheduler
multiple-level queues
natural wait

Key Terms

nonpreemptive scheduling policy

preemptive scheduling policy

priority scheduling

process

process control block (PCB)

process scheduler

process scheduling algorithm

process scheduling policy

process status

round robin

shortest job next (SJN)

shortest remaining time (SRT)

task

thread