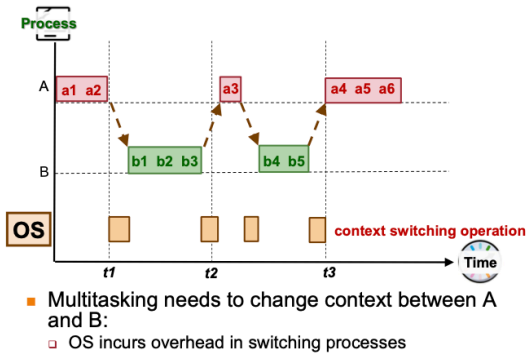
CS2106 Live Class Lec2a: Process management in Unix

$_{/}|/|U_Ch@NgRu!$

May 10, 2021

1 Interleaved execution(Context switch)



processes can be run on multiple CPU(multiple queue of processes) or just one CPU

2 Scheduling problem

If ready-to-run process is more than available CPUs, which should be chosen to run?

2.1 Scheduler

Part of the OS that makes scheduling decision

2.2 Scheduling algorithm

The algorithm used by scheduler

IMPORTANTThe way to schedule is influenced by **scheduling algorithm** and **scheduling algorithms**

3 Process behaviors

3.1 CPU-Activity:

Computation(Number crunching)
Compute-Bound Process spend majority of its time for CPU-activity

3.2 IO-Activity

Requesting and receiving sercive from I/O devices
IO-Bound Process spend majority of its time for IO-activity
ps: In some case, the processes with faster I/O may spend more time on waiting queue, because process complete I/O fast, it joins the queue earlier, leading to longer queue and so process spend more time waiting in queue in A then in B

4 Processing environment

Three categories:

1. **Batch Processing:** No user: No interaction required, No need to be responsive
2. **Interactive**(or Multiprogramming): With active user interacting with system; Should be responsive, consistent in response time
3. **Real time processing:** Have deadline to meet; Usually periodic process

5 Criteria for Scheduling Algorithms

5.1 Fairness

Should get a fair share of CPU time

1. On a per user basis
2. On a per process basis

Also mean no starvation

5.2 Balance

All parts of the computing system should be utilized

6 Two types of scheduling policies

Defined by when scheduling is triggered

6.1 Non-preemptive (Cooperative)

A process stayed scheduled (in running state) until it blocks or give up the CPU voluntarily voluntarily

6.2 Preemptive

1. A process is given a fixed time quota to run(possible to block or give up early)
2. At the end of the time quota, the running process is suspended by the scheduler(Another process get picked if available)

7 batch processing system

1. No user interaction
2. Non-preemptive scheduling is predominant

7.1 Criteria for batch process

7.1.1 Turnaround time

Total time taken, i.e. finish-arrival time
Related to waiting time: time spent waiting for CPU

7.1.2 Throughput

Number of tasks finished per unit time
i.e., the rate of task completion

7.1.3 CPU utilization

Percentage of time when CPU is working on a task

8 Batch process: FCFS

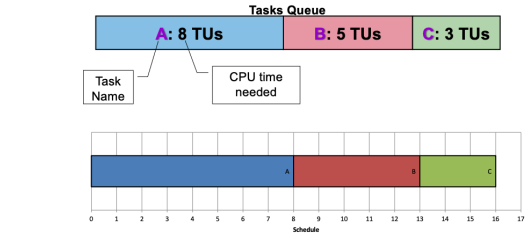
8.1 General Idea

1. Tasks are stored on a First-In-First-Out (FIFO) queue based on arrival time

2. Pick the first task in queue to run until The task is done OR the task is blocked
3. Blocked task is removed from the FIFO queue, When it is ready again, it is placed at the back of queue

8.1.1 Guaranteed to have no starvation

The number os tasks in front of task X in FIFO is always decrease(task X will get its chance eventually)



- The average total waiting time for 3 tasks
  - (0 + 8 + 13)/3 = 7 time units

In the case above, the turnaround time is too long

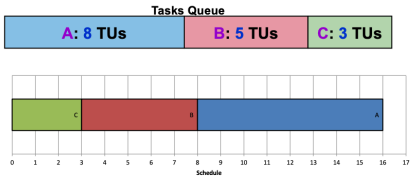
8.2 Convoy effect

If the first task(task A) is CPU-bound while the followed tasks X1,X2,X3,... are are IO-bound

1. The followed x tasks will be kept in ready queue while the IO devide is idling
2. Later, the task A may be blocked on IO because the Xi tasks excute so quickly and occupied IO while the CPU is idling

9 Batch process: Shortest Job First: SJF

Select task with the smallest total CPU time
Needs to predict **total CPU time**
Starvation is possible: Biased towards short jobs; Long job may never get a chance!



- The average total waiting time for 3 tasks
  - (0 + 3 + 8)/3 = **3.66** Time Units
- Can be shown that SJF **guarantees** smallest average waiting time

9.1 Prediction based on previous CPU-Bound phases

$Predicted_{n+1} = \alpha * Actual_n + (1 - \alpha) * Predicted_n$

1. $Actual_n$= The most recent CPU time consumed
2. $Predicted_n$= The past history of CPU time consumed
3. α= Weight placed on recent event or past history
4. $Predicted_{n+1}$= Latest prediction

9.1.1 A scenario where the SJF may fail

The first round of execution:

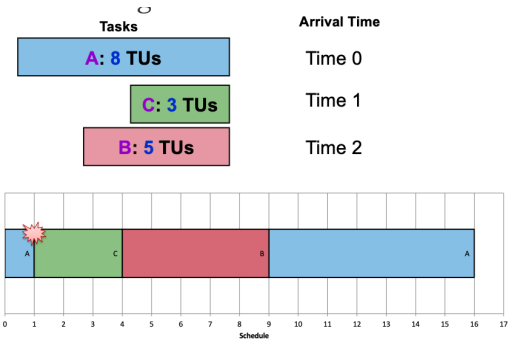
1. A: 1s
2. B: 2s
3. C: 10h
4. D: 3s

the D comes just after C, and it did not even get the chance to run for the evaluation, because the C occupies the CPU

10 Batch process:Shortest remaining time: SRT

10.1 General idea

Use remaining time and it's **preemptive** to solve the problem above
Select job with shortest remaining time(or expectation thereof)
IMPORTANT: New job with shorter remaing time can preempt currently running job
Provide good service for short job even when it arrives late



11 Criteria for interactive environment

11.1 Response time

Time between request and response by system

11.2 Predictability

Variation in response time, lesser variation== more predictatable

12 periodic scheduler invocation

12.1 timer interrupt

Interrupt that goes off periodically based on **hardware clock**
OS ensure timer interrupt cannot be intercepted by any program(Timer interruot handler invokes **scheduler**)

12.2 Interval of timer interrupt(ITI)

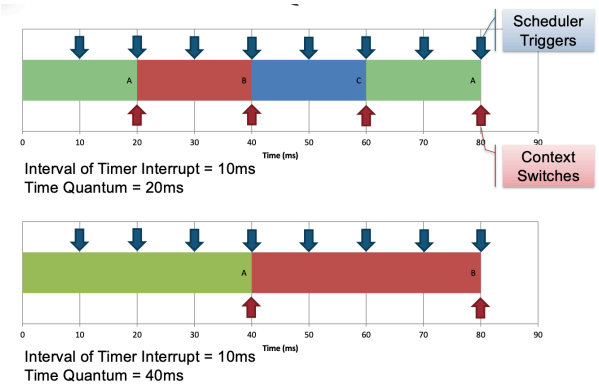
1. scheduler is triggered every timer interrupt
2. Typical values(1ms to 10 ms)

12.3 Time quantum

1. Execution duration given to a process
2. could be constant or variable among the processes
3. must be multiples of interval of timer interrupt(Time quantum=n*ITI)

4. large range of values(commonly 5ms to 100 ms)

ITI vs Time quantum



for the first image, at 10ms, there will be a time interrupt, but it will be ignored by the time scheduler; At 20ms, there is a decision for the scheduler to switch or not
The second image used the same ITI but different time quantum
There is a trade-off for the length of time of quantum

1. too long: like no time quantum(Bigger CPU utilization but longer waiting time)
2. too short: the context-switch takes too much time

13 Scheduling algorithms for interactive algorithm

1. Round robin(RR)
2. Priority based
3. multi-level feedback queue
4. lottery scheduling

14 Round robin(RR)

14.1 General idea

1. Tasks are stored in a FIFO queue
2. Pick the first task from queue front to run until(A fiexed time quantum elapsed(and the task will be blocked); the task gives up the CPU voluntarily; the task blocks)

3. the task is then palced at the end of queue to wait for another turn(blocked will be move to the other queue to wait, when it is ready again, will be put at the end of the ready queue)

14.2 Characters

1. Like preemptive version of FCFS
2. No starvation(Response time guarantee) time before a task is served is bound by (n-1)q, where q is a time quantum
3. Timer interrupt needed(For scheduler to check on quantum expiry)
4. time quantum duration is a trade-off

14.3 When the RR is poor than FIFS

1. When the job lengths are all the same and much greater than the time quantum, RR performs poorly in average turnaround time, because the context-switch takes time
2. When there are many jobs and the job lengths exceed the time quantum, RR results in reducing throughput due to the more time taking for context-switch

IMPORTANT: reducing the time quantum will necessarily increase the turn around time(more time for context-switch) and decrease through put but improve the initial response time and ; Increasing the time quantum will be on the contrary

15 Interactive process: Priority Scheduling

15.0.1 General idea

Some processes are more important than others(Not just arrival time)(Assign a priority value to all tasks and select task with the highest priority)

15.1 Two types of priority scheduling

1. Preemptive version: Higher priority process can preempts running process with lower priority
2. Non-preemptive version: Late coming high-priority process has to wait for next round of scheduling

15.2 The problem of priority scheduling

How do we know which tasks should be signed with higher priority

15.2.1 Should the CPU-bound process be given higher priority for IO

1. yes:CPU processes can be given higher priority for I/O so they may return to waiting for the CPU, decreasing overall turnaround time at the expense of response time of I/O-bound processes
2. no: the algo should maximise responsiveness of IO-bound processes

15.3 priority inversion

When a higher-priority process needs to read or modify kernel data that are currently being accessed by a lower-priority process —or a chain of lower-priority processes. Since kernel data are typically protected with a lock, the higher-priority process will have to wait for a lower-priority one to finish with the resource. The situation becomes more complicated if the lower-priority process is preempted in favor of another process with a higher priority. e.g

Code A	Code B

wait(S);
<do some work>
signal(S);
	<heavy computation>

If a high priority task H and a low priority task L run code A, then it is possible that the L enter the critical section but get pre-empted by H before finishing its work. So, L still "holds" the semaphore, preventing H from executing. At this time, if a medium task M enter the system to run code B, it will get picked over H despite having a lower priority.

16 Interactive process: Multi-Level Feedback Queue

16.1 Basic rule

1. If Priority(A) > Priority(B) → A runs
2. If Priority(A) == Priority(B) → A and B runs in RR

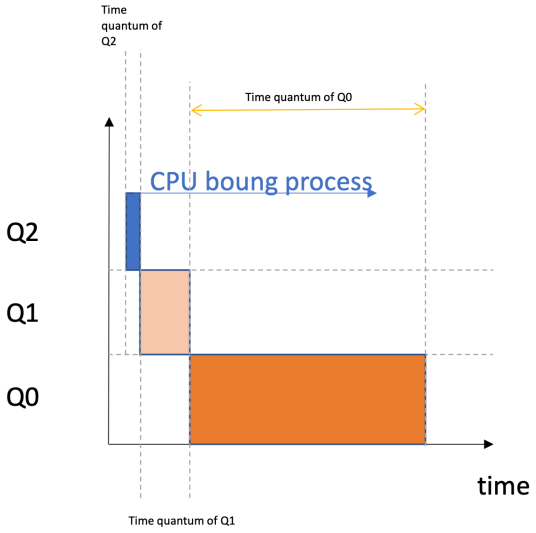
16.2 Priority setting/changing rules

1. New job → highest priority
2. If a job fully utilized its time slice → priority reduced(lower priority means longer quantum time)
3. If a job give up/block before it finishes the time slice → priority retained

16.3 characters

1. Adaptive: learn the process behavior automatically
2. Seeks to minimize both:
   (a) response time for interactive and IO-bound process
   (b) turnaround time for CPU-bound proccess

16.4 Scenario regarding MLFQ



1. A process with a lengthy CPU-intensive phase followed by I/O- intensive phase.
   The process can sink to lowest priority during the CPU-intensive phase. With the low priority, the process may be starved during the IO phase(while the IO phase does not actually take much CPU time)
   way to sove: (Rule –Timely boost) All processes in the system will be moved to the highest priority level periodically.
2. A process repeatedly gives up CPU just before the time quantum lapses.
   If it gives up every time just before the time quantum, the process will remian on the highest priority, which makes it get a lot of CPU time even if it is actually a CPU hogger
   way to solve: The CPU usage of a process is now accumulated across time quanta. Once the CPU usage exceeds a single time quantum, the priority of the task will be decremented.

16.5 Implementation of MLFQ

On of the best way to learn a algorithm is to write it. Because the MLFQ is one of the most complex one, here I implement this:

Consider the standard MLFQ scheduling algorithm with 3 levels (0 = Lowest priority, 2 = Highest priority). The tasks are given different time quantum according to their priority (Priority 0 = 8 time units, priority 1 = 4 time units, priority 2 = 2 time units).

Here assmue assume that all tasks are CPU intensive that runs forever (i.e. there is no need to consider the cases where the task blocks / give up CPU).

NOTE: the scheduler function is invoked by timer interrupt that triggets once a time unit

```
// Variable
Process PCB contains: { PID, TQLeft, ... } // TQ = Time Quantum, other PCB info irrelevant.
ReadyQ[3];
/*
array of 3 PCB queues according to the priority level. The queues supports standard
    operations like isEmpty(), insertBack() and removeFront().
*/
RunningTask;// the PCB of the current task running on the CPU.
TimeQuatum[3] //an array of 3 values { 8, 4, 2} corresponding to the time quantum
given to task at a particular priority level.
TempTask // an empty PCB, provided to facilitate context switching.

//Given function
SwitchContext( PCBout, PCBin );
/*
A function that can save the context of the running task in PCBout, then setup the new
    running environment with the PCB of PCBin, i.e. vacating PCBout and preparing for
PCBin to run on the CPU.
*/

//Implementation of MLFQ
RunningTask.TQleft--; //Running Task consume one given time quantum
If ( RunningTask.TQLeft > 0) return; //no need to schedule
// If the below execute, means context switch needed
If ( RunningTask.priority > 0) RunningTask.priority--; //one task exceed the time quantum,
    should decretment its priority
ReadyQ[ RunningTask.priority]; insertBack( RunningTask ); //Insert to the new priority queue
For(i= 2 down to 0) //same logic can be expressed as a 3-level "if-else"
{
   if !ReadyQ[i].empty()
   {
       TempTask; =ReadyQ[i].removeFront();
```

```
        TempTask.priority = i;
        TempTask.TQleft = TimeQuantum[i];
        break;
    }

}
switchContext( TempTask, Running Task);
```

17 Lottery scheduling

17.1 General Idea

1. Give out "lottery tickets" to processes for various system resources
2. Each time, randomly choose a lottery ticket among eligible tickets(The winner is granted the resource)

17.2 Characters

1. It achieves responsive:a newly created process can participate in the next lottery
2. process with higher priority can be given more tickets
3. Good control for parent-child tickets(If the parents get several tickets, parent can give the tickets to child)
4. Each resource can have its own set of tickets(different proportion of usage per resource per task
5. Simple implementation

18 Interleaving problem

In general, a process are in three step:

1. Read data from memory to register
2. Arithmetic computting
3. store the data back to memory from register back to memory

When multiple process write on the same memory and no synchronization involved, there are two reasons that can result in the wrong result

1. On multiple cores, multiple processes execute in parallel and happens to interleave instruction
2. On single core, the step 1,2,3 above of different processes may interleaved, which mauy also result in the wrong answer

19 Responsiveness of scheduling algorithm

19.1 Definition

The responsiveness of a scheduling algorithm refers to how soon can a newly created task receives its first share of CPU time.
If a newly created task T_{new} added into an environment where there are N (N > 0) ready to run tasks.
IMPORTANT: the short waiting/responsive time (i.e. the task gets CPU quickly) is different to bound waiting time(i.e. guaranteed no starvation)

19.2 MLFQ

Good responsive: Every new coming task will be given the highest priority, hence it is responsive

19.2.1 Priority

Good responsive: can set the built in features to enables T_{new} to receive its first share of CPU time as soon as possible

19.2.2 Lottery

Good responsive: new coming task can be given more tickets to make it get the CPU share earlier

19.3 SJF

Fair responsive: the new coming task can be given a perhaps short predict value, so that they may be allocated to a relative advanced position

19.4 SRT

Fair responsive: the reason is the same to SJF

19.5 Round Robin

Fail to be responsive, given an environment with N tasks, T_{new} will be queued at the end and only receive cpu time after n * time_quantum, i.e. much slower than other algorithms discussed above.

19.6 First come first served

Fail to be responsive, even worse than round robin, queue at the end, and must wait all the task to be finished