COSC 3750

Scheduling

Kim Buckner

University of Wyoming

Mar. 01, 2022

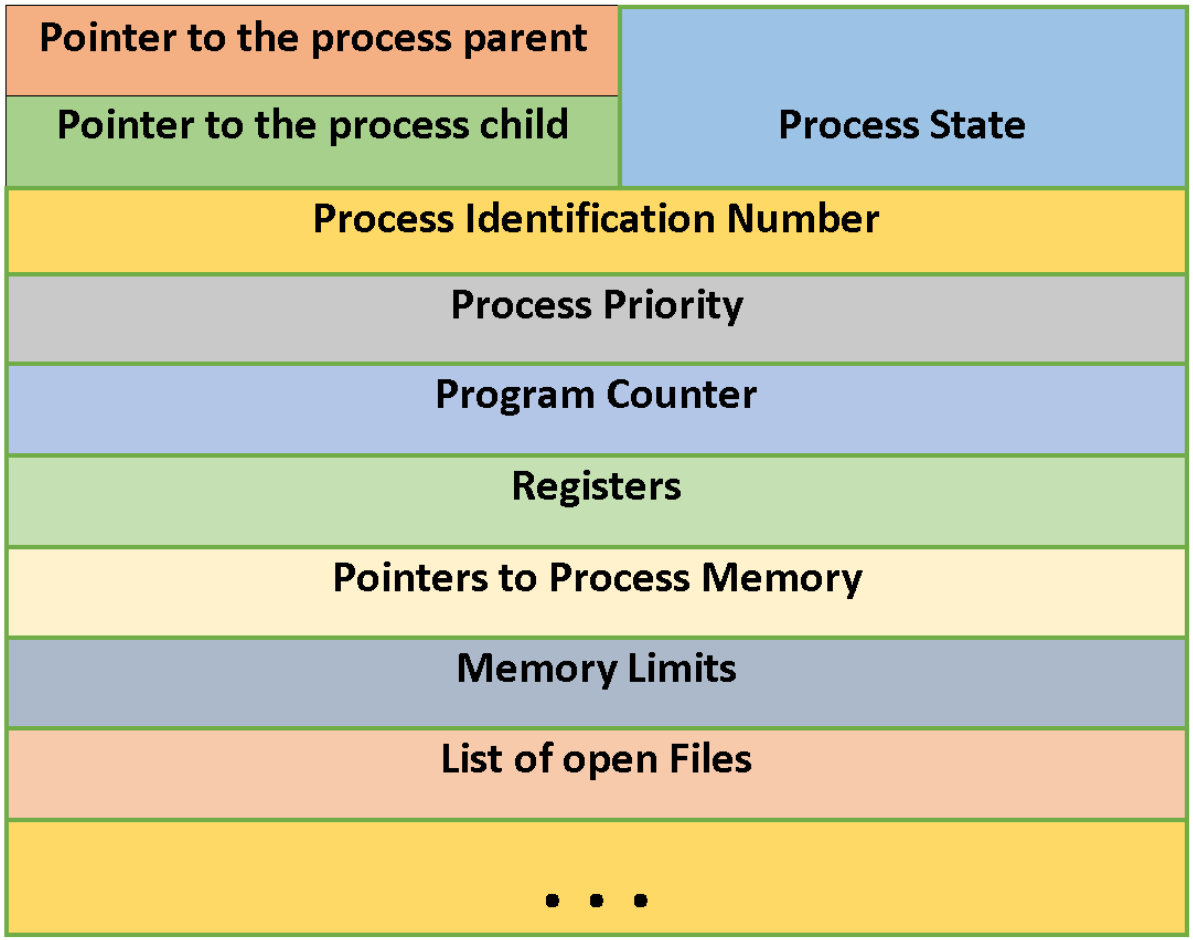
Processes

- A process is an active entity whereas a program is a passive one.
- Every process has a state (distinguished from 'state'):
  - new,
  - running,
  - waiting,
  - ready,
  - or terminated.

(more . . .)

- Every process has a process control block (PCB) containing
  - its status,
  - pid,
  - PC,
  - registers,
  - memory limits,
  - open files,
  - memory map,
  - scheduling info,
  - accounting info, etc.

PCB


Pointer to the process parent
Pointer to the process child
Process State
Process Identification Number
Process Priority
Program Counter
Registers
Pointers to Process Memory
Memory Limits
List of open Files
• • •
created by Notes Jam

CPU Scheduling

- Main idea in a multiprogramming environment: have some process running all the time.
- Process runs until it waits.
- Wait is usually for I/O.

Burst cycle

- Scheduling success depends a cycle–CPU time followed by I/O time
- Time is in "bursts."
- This describes the duration of the portion of the cycle
- I/O bound process have very short (usually) CPU bursts and relatively long I/O bursts.
- Has been a lot of study, durations are usually exponential or hyper-exponential.

CPU Scheduler

- Short-term scheduler selects next process to run after CPU becomes idle.
- Ready queue is not necessarily FIFO.
- Queue generally contains PCBs.

Preemptive scheduling

- The decision to change a process state is made. Then ...
  1. the process switches from running to waiting, or
  2. the process switches from running to ready, or
  3. the process switches from waiting to ready, or
  4. the process terminates.

(more . . .)

- states 1 & 4 require an immediate choice from the ready queue, easy decision.
- 1 & 4 = non-preemptive = Windows 3.1 and Apple Macintosh.
- Otherwise preemptive.
- Preemptive can be costly, i.e. shared data updates being interrupted.

(more . . .)

- What happens when O/S is performing on behalf of process that is preempted?
  - Wait until it completes or
  - back off and cause it to be restarted when process runs again.
- Can of course just disable interrupts but this can be costly and may not scale well to multiprocessor systems

Dispatcher

- Gives control of CPU to a process.
- Switches context.
- Switches to user mode.
- Jumps to proper location to restart.
- Must be fast - dispatch latency can be a problem.

Scheduling criteria

- Lots of different methods have proposed and used.
- CPU utilization should range from 40 to 90 per cent but in real life the computation of this is often misleading.

Some bases (bā-sēz)

- Throughput - work being done
  - number of processes completing per time unit
- Turnaround time - time to run a process from submit to complete
  - sum of wait time, run time, and I/O time.
- Waiting time - process is runnable but not running.

(more . . .)

- Response time - for interactive processes
  - from submit until responding starts but NOT time to output response
- Want to maximize CPU utilization and throughput and minimize waiting, turnaround and response time
- Some suggest minimizing variance in response time rather than reducing average

Scheduling algorithms

FCFS

- FCFS - First-come, First-served. Easily managed with a FIFO queue.
- Average wait time can be quite long.
- Generally not minimal, can be quite long, depends entirely on order of submission.
- Non-preemptive, bad for time-sharing systems.

SJF, shortest-job-first

- Based on last CPU burst.
- Provably optimal.
- But how to predict next burst?
- Usually with an exponential average.
  - $T_n + 1 = at_n + (1 - a)T_n$
  - Where T_n is the history and $0 \leq a \leq 1$ is the weight of the history and the last burst
  - often a $= 1/2$.
  - T_0 can be a constant or the system average, whatever

(SJF more . . .)

- Can be either preemptive or non-preemptive.
- Sometimes called shortest-remaining-time-first if preemptive.

Priority scheduling

- Each process is associated with a priority.
- SJF is a form of this.
- Priorities are usually in a small fixed range:
  - 0-7,
  - 0-4095,
  - but no one can agree if 0 is the highest or lowest.

(more . . .)

- We will use ‘zero is low’ unless we specify otherwise.
- Two types of criteria:
  - internal–time, cpu usage, number of open files,
  - external–importance to someone, like amount being paid for cpu use.
- Can be preemptive or non-preemptive.

(more . . .)

- Main problem is starvation.
- Solution to that is aging, as a process stays in the run queue it keeps getting its priority improved until it finally runs

Round Robin

- Time-sharing systems.
- Preemption added to FCFS.
- Each process is allowed to run for a maximum of 1 time quantum.
- Ready queue is FIFO.
- Either the process will
  - run a quantum and be preempted or
  - it will make a request that causes it to relinquish the queue.

(more . . .)

- Average wait time can be long.
- Each process gets $1/n$ of the CPU time and waits at most $(n-1) * q$ time.
- If the quantum is large, RR becomes FCFS.
- One consideration is the time for a context switch.
- Want time quantum to be large with respect to the context switch.

(more . . .)

- Turnaround time is affected as well but does not necessarily improve with an increase in quantum.
- Best if average CPU burst time is 80% of the time quantum.

Multi-level queue

- Different 'types' of processes have different priority queues.
- Each queue can have a different type of scheduling.
- Must be a scheduling between queues too, say preemptive priority, maybe with aging.

(more . . .)

- Might could time slice the queues with each one having a separate time quantum.
- Or maybe just a percentage of CPU time for any particular queue.

M-L feedback

- Allows process to migrate between queues.
- Uses promotion and demotion.
- Each queue has a different scheduling method.
- etc.