Processor Management

INFO 2603
Platform Technologies 1

Week: 18-Sept-2018

Processor Manager

The **Processor Manager** is responsible for allocating the processor (**CPU**) to execute instructions for incoming jobs.

It is required to monitor the computer's CPU to see if it is busy executing a process or sitting idle as it waits for some other command to finish execution.Generally, systems are more efficient when their CPUs are kept busy

The Process Manager handles each process's transition from one state of execution to another as it moves from the starting queue, through the running state, and then to the finished state.

It does this by keeping track of the status of each **job**, **process**, **thread** etc.

Programs vs Processes vs Threads

A **program** is a passive unit, such as a stored file that resides in secondary memory. It consists of instructions written in a programming language.

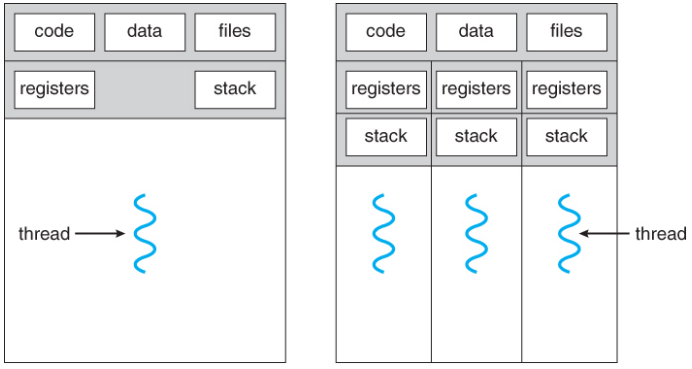
A program or a **job** is usually composed of multiple processes.

A **process** is an active entity that requires a set of resources to perform its function. It consists of instructions written in machine code and resides in main memory. These resources include the CPU and special registers.

A process or a **task** can consist of multiple threads.

A **thread** is created by a process and can be scheduled and executed independently of its parent process.

Single Threaded vs Multithreaded Processes



single-threaded process

multithreaded process

Multithreading

Multithreading allows applications to manage a separate process with several threads of control.

Web browsers use multithreading routinely when one thread retrieves images while another sends and retrieves email. Multithreading can also be used to increase responsiveness in a time-sharing system, increase resource sharing and decrease overhead.

Multithreading Example

If your single-CPU system allows its processes to have a single thread of control, and you want to see a series of pictures on a friend's website, you can instruct the browser to establish one connection between the two sires and download one picture at a time.

However, if your system allows processes to have multiple threads of control (a more common reality), then you can request several pictures at the same time. The browser will then set up multiple connections (using threads) and download several pictures, seemingly at one, even though it is using only one CPU.

Multiprogramming

Multiprogramming requires that a processor be allocated to each job or process for a period of time and then deallocated at an appropriate moment.

Consider if a CPU is shared between two programs or jobs (A and B) each with specific and different instructions. Each step in job A can be called a process. Suppose a higher priority program, Job B, comes along. Job A, which is currently running, will be interrupted in favour of the higher priority one. This is called a context switch. Before being interrupted the last completed instruction in Job A is noted. When the CPU is handed back over to Job A once more (another context switch), the job resumes until either it is finished or deallocated from the CPU again.

Multiprogramming Example

1. Get the input for Job A
2. Identify Job A resources
3. Execute the process
4. Receive the interrupt
5. Perform a context switch to Job B
6. Get the input for Job B
7. Identify Job B resources
8. Execute the process
9. Terminate Job B
10. Perform a context switch to Job A
11. Resume executing the interrupted process
12. Terminate Job A

Multi-Core Technologies

A dual-core or quad-core or multi-core CPU has more than one processing element- sometimes called a core or CPU-on the computer chip.

Multi-Core engineering was driven by the problems caused by nano-sized transistors and their ultra-close placement on a computer chip. Close proximity-> dramatic increase in system performance. Close proximity->excessive heat, current loss, and circuit failure.

Solution: singe chip (one piece of silicon) housing two or more cores. Same physical space but larger number of smaller sized processors. -> less heat and less current loss

Scheduling Sub-managers

The Processor Manager is composite of two sub-managers:

- Job Scheduler - high level scheduler
- Process Scheduler - low level scheduler

The two schedulers work together to ensure fair and efficient use of the CPU by programs and processes.

Job Scheduler

The Job Scheduler selects jobs from a queue of incoming jobs and places them in the process queue based on each job's characteristics.

One goal is to put jobs (still residing on disk) in a sequence that best meets the designers or administrator's goals. e.g. using system resources as efficiently as possible.

Another goal is to create an order for the incoming jobs that has a balanced mix of I/O interaction and computation requirements, thus, balancing the system's resources.

Essentially, it aims to keep most of the components of the computer systems busy most of the time.

Process Scheduler

The Process Scheduler takes over the jobs that have been accepted by the Job Scheduler to run (Ready queue).

If threads are supported, the Process Scheduler handles them.

The Process Scheduler

- determines which processes will get the CPU, when, and for how long based.
- decides what to do when processing is interrupted
- determines which queues a job should be allocated to during its execution
- recognises when a job is finished or should be terminated

I/O Bound vs CPU Bound Jobs

Most programs alternate between CPU cycles and I/O cycles. The duration and frequency of CPU cycles vary from program to program.

I/O Bound Jobs: have long input or output cycles and brief CPU cycles eg. printing a series of documents

CPU Bound Jobs: have long CPU cycles and brief I/O cycles eg. finding the first 300,000 prime numbers

Example

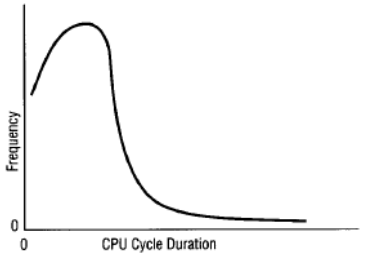
1. Ask the user for the first number
2. Retrieve the first number that's entered: Input #1
3. Ask the user for the second number
4. Retrieve the second number that's entered: Input #2
5. Add the two numbers: Input #1 + Input#2
6. Divide the sum from the previous calculation by 2 to get the average of the two inputs: (Input #1 + Input#2)/2 = Output#1
7. Multiply the result of the previous calculation by 12 to get the average for the year: Output#1 * 12 = Output#2
8. Print the calculated average: Output#1
9. Print the average for the year: Output#2
10. End

Brief I/O Cycle (1–4)

Brief CPU Cycle (5–7)

Brief I/O Cycle (8–9)

CPU Cycle Distributions

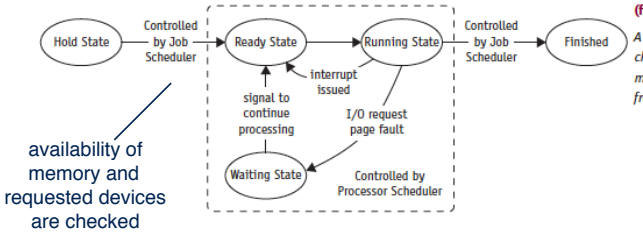


(figure 4.1)

Distribution of CPU cycle times. This distribution shows a greater number of jobs requesting short CPU cycles (the frequency peaks close to the low end of the CPU cycle axis), and fewer jobs requesting long CPU cycles.

A middle-level scheduler might be engaged to remove active jobs from memory to reduce the degree of multiprogramming (system becomes overloaded). This allows other jobs to be completed faster.

Jobs and Process States

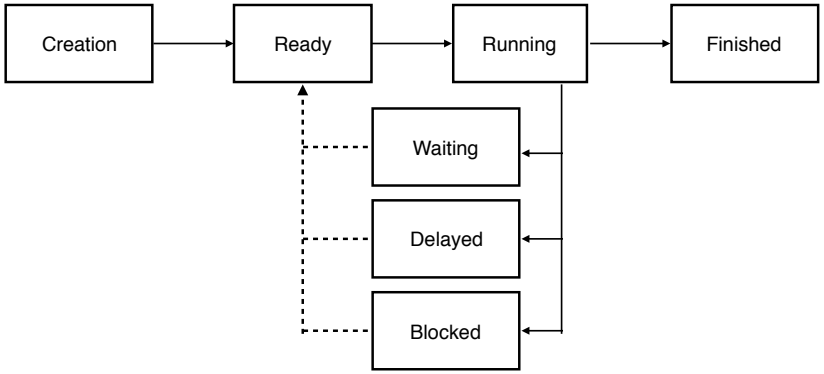


(figure 4.2)

A typical job (or process) changes status as it moves through the system from HOLD to FINISHED.

Thread States

Just as for processes, threads have states: Creation, Ready, Running, Waiting, Delayed, Blocked, Finished.



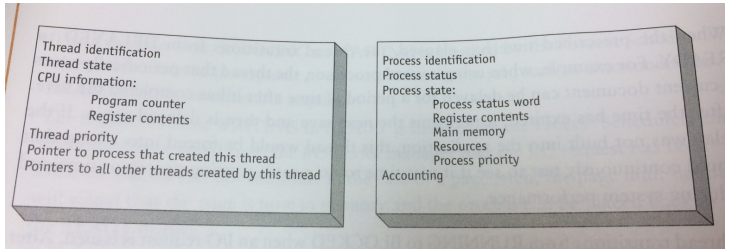
Control Blocks

Each process in the system is represented by a data structure called a **process control block (PCB).** Threads have a similar data structure called a **thread control block**.

The **PCB** is created when the **Job Scheduler** accepts the job, and is updated as the job progresses from start to end.

The **TCB** is created by the **Process Scheduler** to track a thread's progress from beginning to end.

Control Blocks



Comparison of a TCB (left) vs a PCB (right)

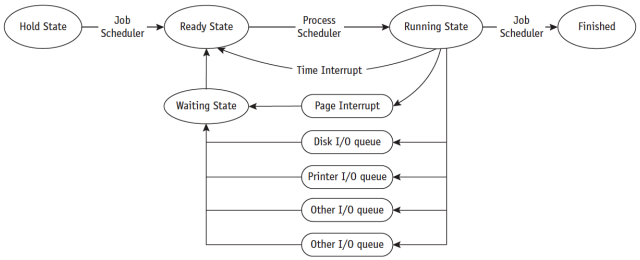
Chpt 4, Page 114: Understanding Operating Systems, McHoes & Flynn, 2017

Queues

Queues use control blocks to track jobs. The control blocks contain all of the data needed by the OS to manage processing of the job.

As the job moves through the system, its progress is noted in the control block.

Control Blocks and Queuing



(figure 4.4)

Queuing paths from HOLD to FINISHED. The Job and Processor schedulers release the resources when the job leaves the RUNNING state.

Scheduling Policies and Algorithms

A single processor can be shared by several jobs or several processes, if and only if, the OS has a scheduling policy and a scheduling algorithm. These are used to determine when the CPU has to stop working on one job and proceed to another.

These algorithms can be governed by several factors such as job priority or length of time required to complete the job for example.

A timing mechanism is used by the Process Scheduler to periodically interrupt processes when a predetermined slice of time has expired.

An I/O request is called a natural wait in multiprogramming environments.

Scheduling Algorithms

- First-Come, First-Served: the earlier a job arrives, the soon it is served. No external interrupts. Common in batch systems. Unacceptable for interactive systems (need a quick response time)
- Shortest Job Next: chooses jobs based on length of CPU cycle time required. Needs the times in advance - good for batch systems; doesn't work in interactive systems unless times known in advance.No external interrupts
- Priority Scheduling: preferential treatment to important jobs based on priority. Jobs aren't interrupted until completed or a natural wait occurs. No external interrupts
- Shortest Remaining Time: processor allocated to job closest to completion. Needs advance knowledge of times. High CPU overhead - frequent monitoring of ready queue. Allows interrupts
- Round Robin: time slices of fixed lengths are equally shared among active processes. Allows interrupts.

Scheduling Algorithms

Algorithm	Policy Type	Best for	Disadvantages	Advantages
FCFS	Nonpreemptive	Batch	Unpredictable turnaround times	Easy to implement
SJN	Nonpreemptive	Batch	Indefinite postponement of some jobs	Minimizes average waiting time
Priority scheduling	Nonpreemptive	Batch	Indefinite postponement of some jobs	Ensures fast completion of important jobs
SRT	Preemptive	Batch	Overhead incurred by context switching	Ensures fast completion of short jobs
Round robin	Preemptive	Interactive	Requires selection of good time quantum	Provides reasonable response times to interactive users; provides fair CPU allocation
Multiple-level queues	Preemptive/ Nonpreemptive	Batch/ interactive	Overhead incurred by monitoring of queues	Flexible scheme; counteracts indefinite postponement with aging or other queue movement; gives fair treatment to CPU-bound jobs by incrementing time quantums on lower-priority queues or other queue movement

(table 4.1)

Comparison of the scheduling algorithms discussed in this chapter.

Context Switching

Context switching is the act of saving a jobs processing information in its PCB so that the job can be swapped out of memory. The next job's processing information is loaded in the appropriate registers so that the CPU can process it.

Context switching is required by some of the scheduling algorithms which allow interrupts.

Managing Interrupts

Interrupts can be generated by:

- Illegal arithmetic operations: divide by 0, overflow or underflows from floating point operations
- Illegal job instructions:
  - attempts to access protected, non-existent memory locations
  - attempts to use an undefined operation code
  - attempts to operate on invalid data
  - attempts to make unauthorised system changes (time quantum for scheduling algorithms)

Managing Interrupts

The Interrupt Handler typically follows this sequence when an error is not recoverable:

1. The type of interrupt is described and stored. Passed on to the user
2. The state of the interrupted process is saved (value of PC, mode specification, register contents)
3. The interrupt is processed: message sent to user, program halted, resources released, job exits the system
4. The processor resumes normal operation

For fatal, nonrecoverable errors, the job terminates at step 3.

Reading Resources

Understanding Operating Systems: Ann McIver McHoes, Ida M. Flynn. 8th Edition. 2017: Chapter 4