Control Systems Optimization

Igor Wojnicki

AGH – Univeristy of Science and Technology

2010

Outline

1. Multiple pocessess - why?

Scheduler

- *Scheduler* – a part of the OS which decides which process should be run next.
    - Not always part of the OS.
- *Scheduling algorithm* – an algorithm used by the scheduler.

Scheduling Matters

- Timesharing: multiple users/processes waiting for service, the algorithm has a tremendous impact on the perceived performance of the system.
  - Scheduling matters even on simple PCs !!!
  - It matters even more for Control Systems.

Aim of Scheduling

- Response time
- Throughput
- Processor efficiency

Types of Scheduling

- Long term – a decision to add to a pool of processes to be executed.
- Medium term – a decision to add to a pool of processes in main memory.
- Short term – which process to execute.
- I/O – which process's I/O request should be satisfied.

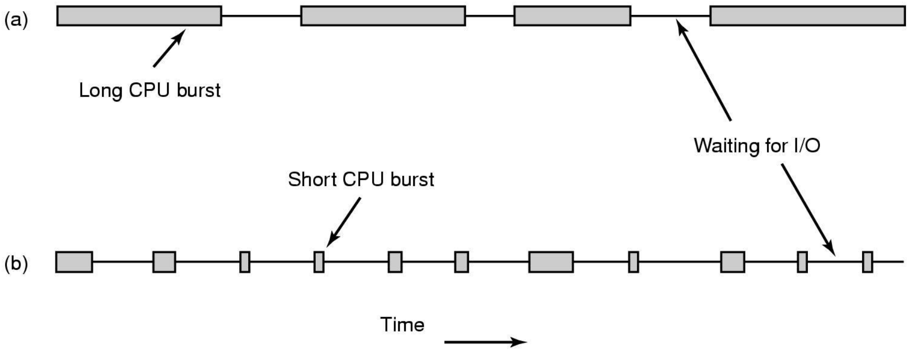
Short Term Scheduling

- Known as the dispatcher
- Executes most frequently
- Invoked when an event occurs
    - Process creation/termination
    - Clock interrupts
    - I/O interrupts
    - Operating system calls
    - Blocking/Unblocking

Short-Term Scheduling Criteria

- User-oriented
  - Response Time – Elapsed time between the submission of a request until there is output.
- System-oriented
  - Effective and efficient utilization of the processor
- Performance-related
  - Quantitative
  - Measurable such as response time and throughput
- Not performance related
  - Qualitative

Process Behavior: CPU-bound vs. I/O-bound



- The key value is the length of CPU burst, not I/O burst.
- Bursts of CPU usage alternate with periods of I/O wait
  - a CPU-bound process
  - an I/O bound process
- The faster CPU the more I/O bound a process is.

Decision Mode

- Nonpreemptive
  - Once a process is in the running state, it will continue until it terminates or blocks itself for I/O
- Preemptive
  - Currently running process may be interrupted and moved to the Ready state by the operating system
  - Allows for better service since any one process cannot monopolize the processor for very long
  - Sometimes not needed in RT systems?

Scheduling Algorithm Goals

- All systems
  - Fairness – giving each process a fair share of CPU.
  - Policy enforcement – seeing that stated policy is carried out.
  - Balance – keeping all parts of the system busy.
- Interactive systems
  - Response time – respond to requests quickly.
  - proportionality – meet users' expectations.
- Real-time systems
  - Meeting deadlines – avoid losing data.

Round Robin Scheduling // aka. Time Slicing

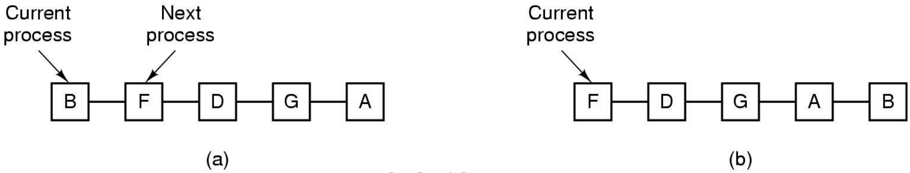


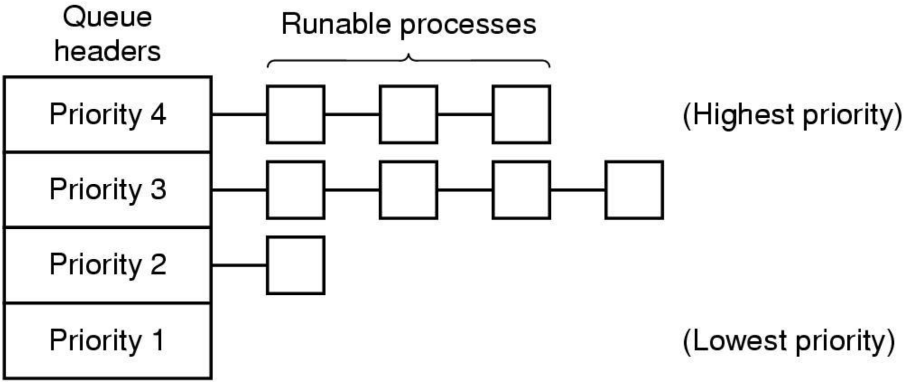
Figure: Round Robin Scheduling.

- List of runnable processes
- List of runnable processes after B uses up its quantum

Priorities

- Scheduler will always choose a process of higher priority over one of lower priority
- Have multiple ready queues to represent each level of priority
- Lower-priority may suffer starvation
  - allow a process to change its priority based on its age or execution history

Priority-based Scheduling



RT OS Scheduling

- Preemptive vs Nonpreemptive
- Preemptive Processes and Preemptive kernel

Deadlock Real Life

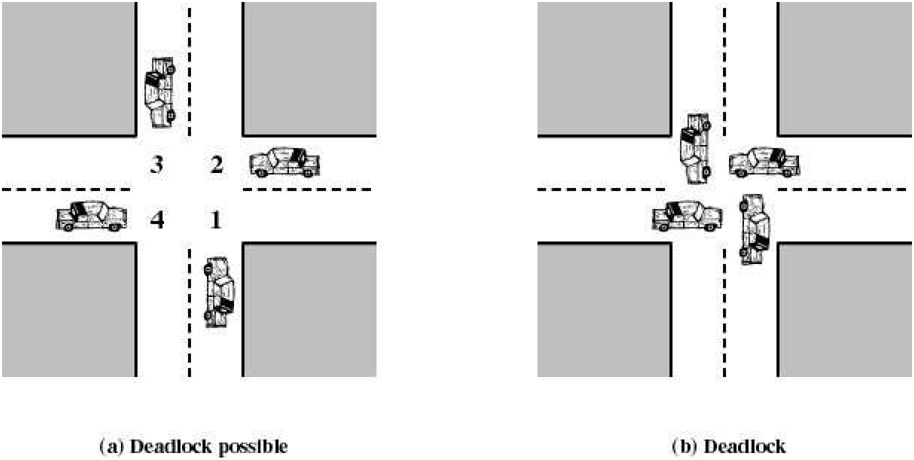


Figure 6.1 Illustration of Deadlock

Deadlock IT

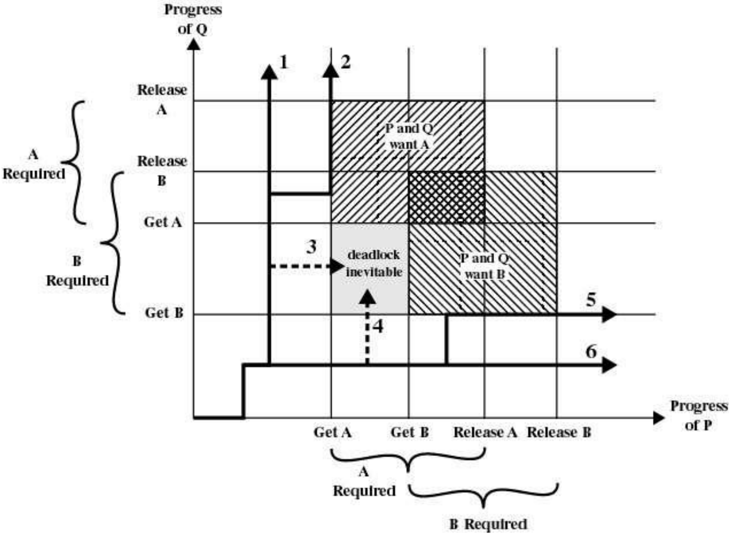


Figure 6.2 Example of Deadlock [BACO08]

Introduction to Deadlocks

- Formal definition : A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause
- Usually the event is release of a currently held resource
- None of the processes can:
  - run
  - release resources
  - be awakened

Starvation

- A process is perpetually denied necessary resources.
- It starves to death not being able to finish its task.

Starvation and Priorities

- A high priority process A will run before a low priority process B.
- If the high priority process (process A) never blocks, the low priority process (B) will (depending on the scheduling algorithm) never be scheduled.
- It will experience *starvation*.
- If there is an even higher priority process X, which is dependent on a result from process B, then process X might never finish, even though it is the most important process in the system.
- This condition is called a *priority inversion*.
- Modern scheduling algorithms normally contain code to guarantee that all processes will receive a minimum amount of each important resource (most often CPU time) in order to prevent any process from being subjected to starvation.

Semaphores C Example I

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <wait.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

union semun {
        int              val;
        struct semid_ds *buf;
        unsigned short   *array;
};
```

Semaphores C Example II

```
int main(void)
{
  int pid,semid;

  struct sembuf oper;

  if ((semid=semget(IPC_PRIVATE,1,0777))<0){
    perror("semaphore allocation");
    exit(1);
  }
  semctl(semid,0,SETVAL,(union semun)1);

  /* semaphore down */
  printf("parent sem down\n");
```

Semaphores C Example III

```
oper.sem_num=0; oper.sem_op=-1; oper.sem_flg=0;
semop(semid,&oper,1);
printf("parent sem down ok\n");
pid=fork();
if (pid==0){
  /* child */

  /* semaphore down */
  printf("child sem down\n");
  oper.sem_num=0; oper.sem_op=-1; oper.sem_flg=0;
  semop(semid,&oper,1);
  printf("child sem down ok\n");

  printf("child doing its job\n"); sleep(2); /* do the job */
```

Semaphores C Example IV

```
    /* semaphore up */
    printf("child sem up\n");
    oper.sem_num=0; oper.sem_op=1; oper.sem_flg=0;
    semop(semid,&oper,1);

    printf("child exit\n");
    exit(0);
  }
  printf("parent doing its job\n"); sleep(2); /* do the job */

  /* semaphore up */
  printf("parent sem up\n");
  oper.sem_num=0; oper.sem_op=1; oper.sem_flg=0;
  semop(semid,&oper,1);
```

Semaphores C Example V

```
  wait(0);
  /* remove the semaphore */
  semctl(semid,0,IPC_RMID,(union semun)0);
  printf("parent exit\n");
  return 0;
}
```