Operating Systems 2

Processes

Introduction

- What is a process….
- The process scheduler
- The process control Block; elements of the PCB
- Creating process in Unix/Linux
  - fork() command
  - Wait() command
  - exec commands
- Multiprocessing system: scheduling; interprocess communication (synchronisation, asynchronous)

Some revision

- Simple system
  - Single user
  - One processor: busy only when executing the user's job or system software

- Multiprogramming environment: multiple *processes* competing to be run by a single CPU
  - Requires fair and efficient CPU allocation for each job

Definitions

- Processor (CPU)
  - Performs calculations and executes programs
- Program (job)
  - Inactive unit, e.g., “an application stored on a secondary storage disk”
  - Unit of work submitted by the user
- Process (task)
  - A single instance of an executable program
  - Active entity that Requires resources (such as processor, special registers, its own memory space, and context data or process control box (PCB)) to perform its function

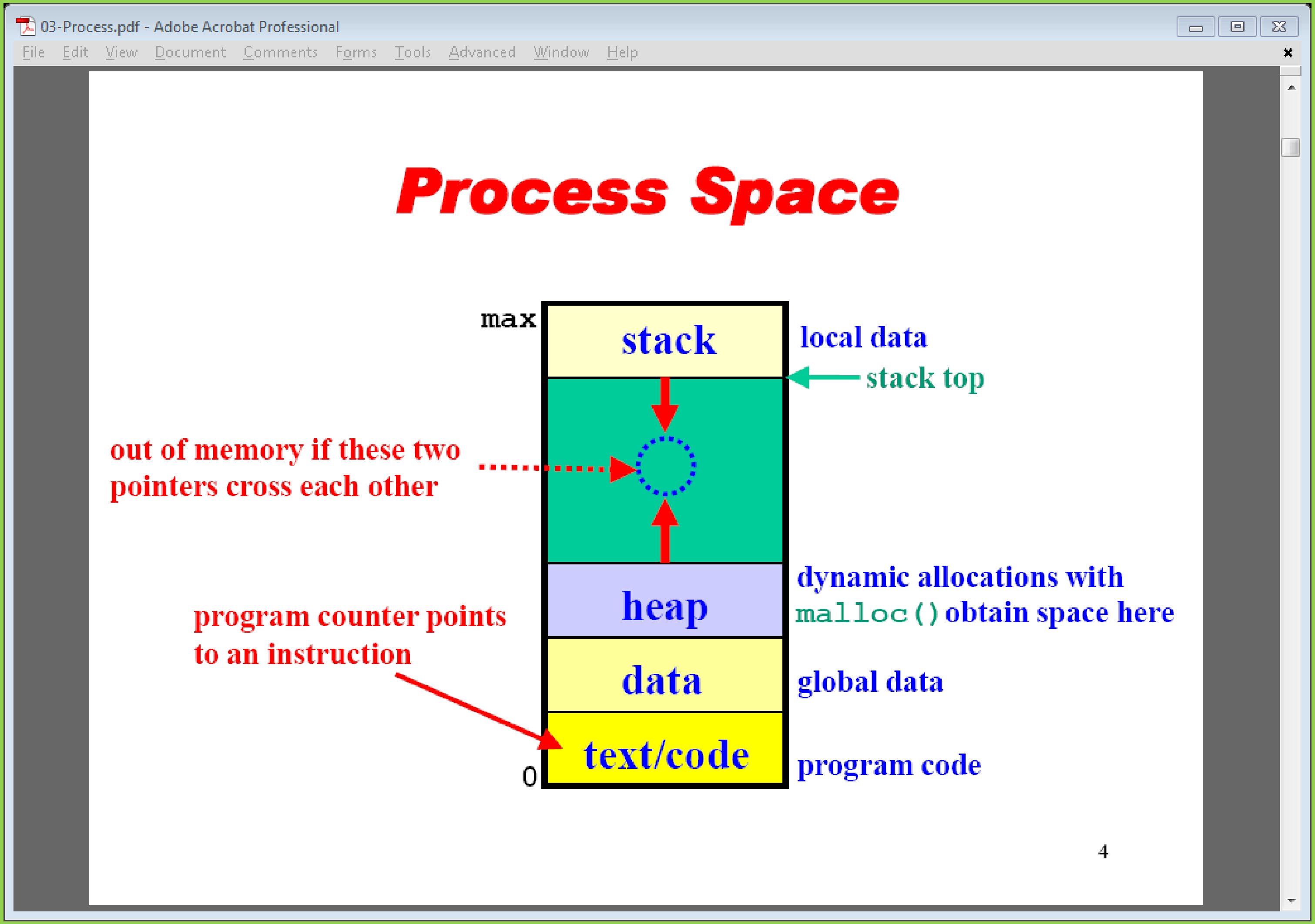
Definitions (cont'd.)

- Thread (of control)
  - Portion of a process; a process can have *multiple* threads of control
  - A process is equivalent to a program while threads are equivalent to modules/methods within the program
  - A thread is executed "independently" *within* the parent process;

Definitions

- Multiprogramming
  - Processor allocated to each job/process/thread for a time period
  - E.g. two or more programs (jobs). The jobs are independent of one another.
  - Executing job 1 and job 2 is in a **queue**. At an appropriate time (i/o instruction) an interrupt is evoked; and the position of execution is noted: the register contents including the program counter are updated.
  - Then job 2 is executed and either at an interrupt or when finished job 1 begins from point where it stopped executing
  - In a single processor the jobs are interleaved (*run concurrently*)
  - In multiprocessor system can be run in parallel

Process Space



Scheduling Submanagers (cont'd.)

- Job Scheduler functions
  - Selects incoming job from queue
  - Places in process queue
  - Decides on job initiation criteria
    - Process scheduling algorithm and priority (priority queues)
- Goal
  - Sequence jobs
    - Efficient system resource utilization
  - Balance I/O interaction and computation
  - Keep most system components busy most of time

Process Scheduler

- Process Scheduler functions
  - Determines process to get CPU resource
    - When and how long
  - Decides interrupt processing
  - Determines queues for process movement during execution
  - Recognizes job conclusion
    - Determines process termination
- Lower-level scheduler in the hierarchy
  - Assigns CPU to execute individual actions: jobs placed on READY queue by the Job Scheduler

Process Scheduler (cont'd.)

- Exploits common computer program traits
  - Programs alternate between two cycles
    1. CPU
    2. I/O cycles
- General tendencies
  - I/O-bound job
    - Many brief CPU cycles and long I/O cycles (printing documents)
  - CPU-bound job
    - Many long CPU cycles and shorter I/O cycles (a complex maths calculation)

Job and Process States

- Status changes: as a job or process moves through the system
  - HOLD and placed in a queue (sometimes a priority queue
  - READY (job begins processing if enough resources available)
  - WAITING (can not continue until specific resource available e.g. i/o request
  - RUNNING: processing
  - FINISHED

Process Management

- The entire *state of the process* at any instant is contained in its *context (process control block)*

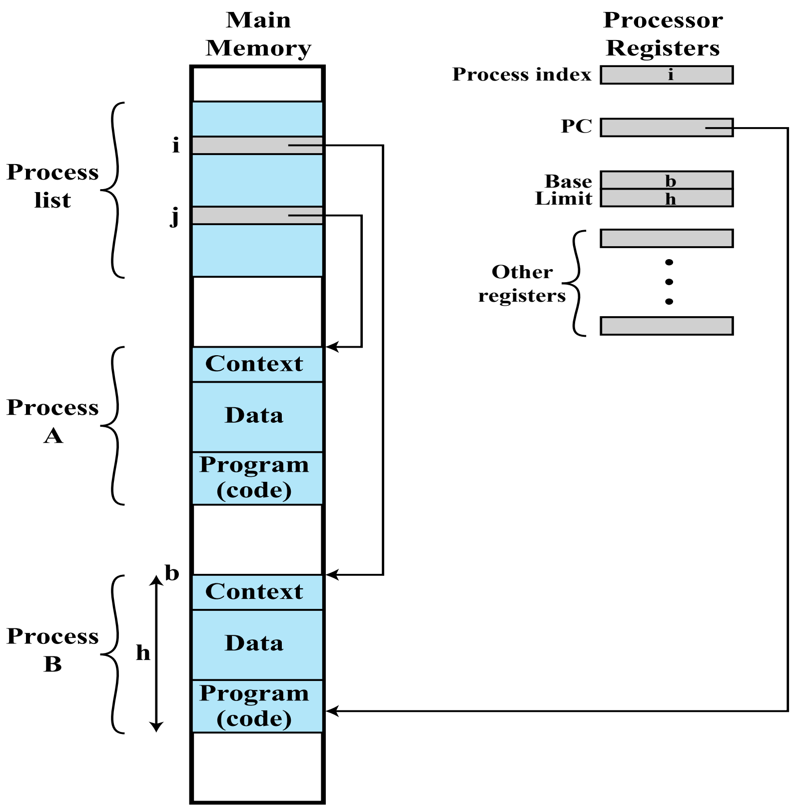
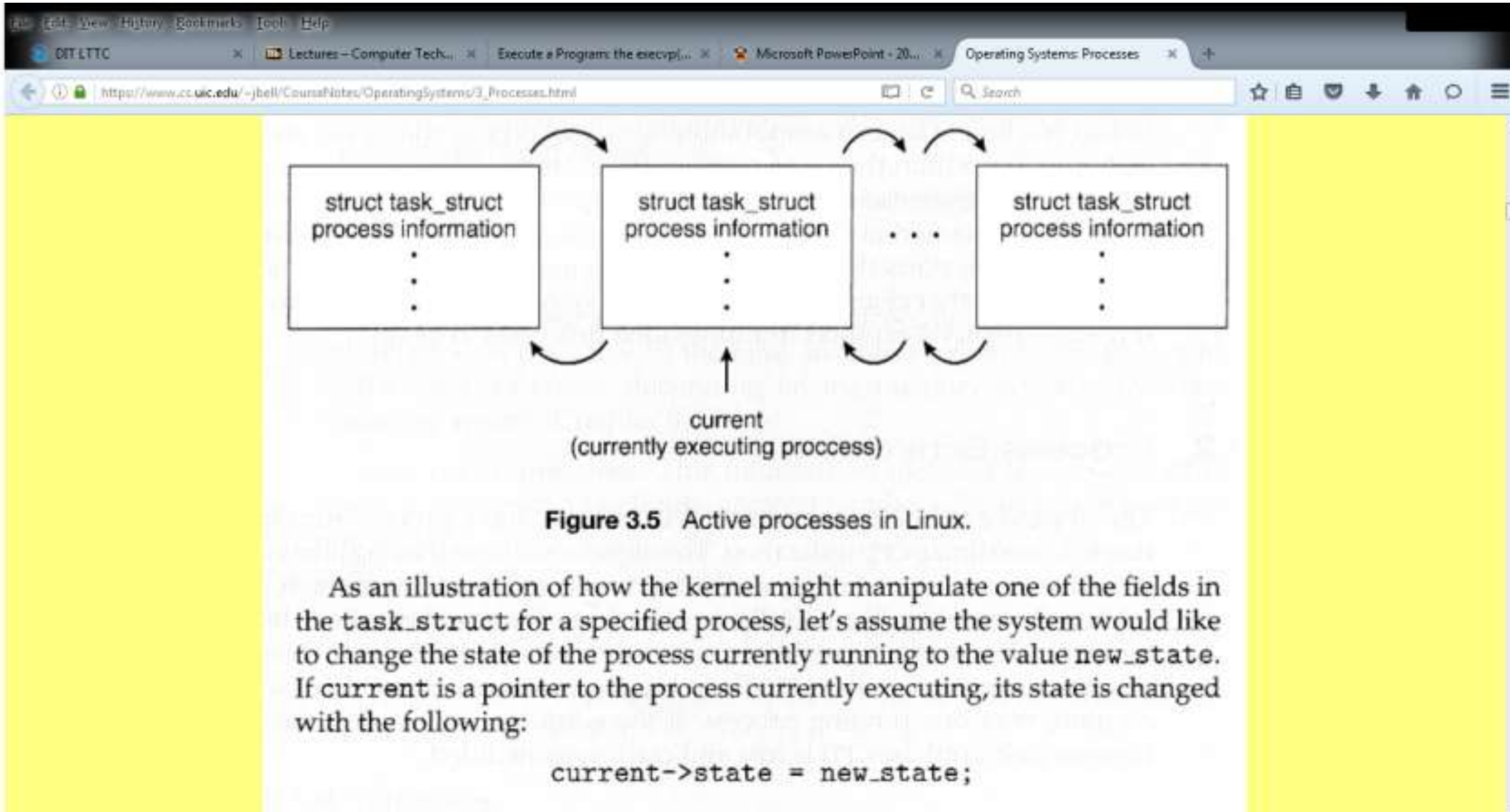


Figure 2.8 Typical Process Implementation

Fields of a process control block (PCB)

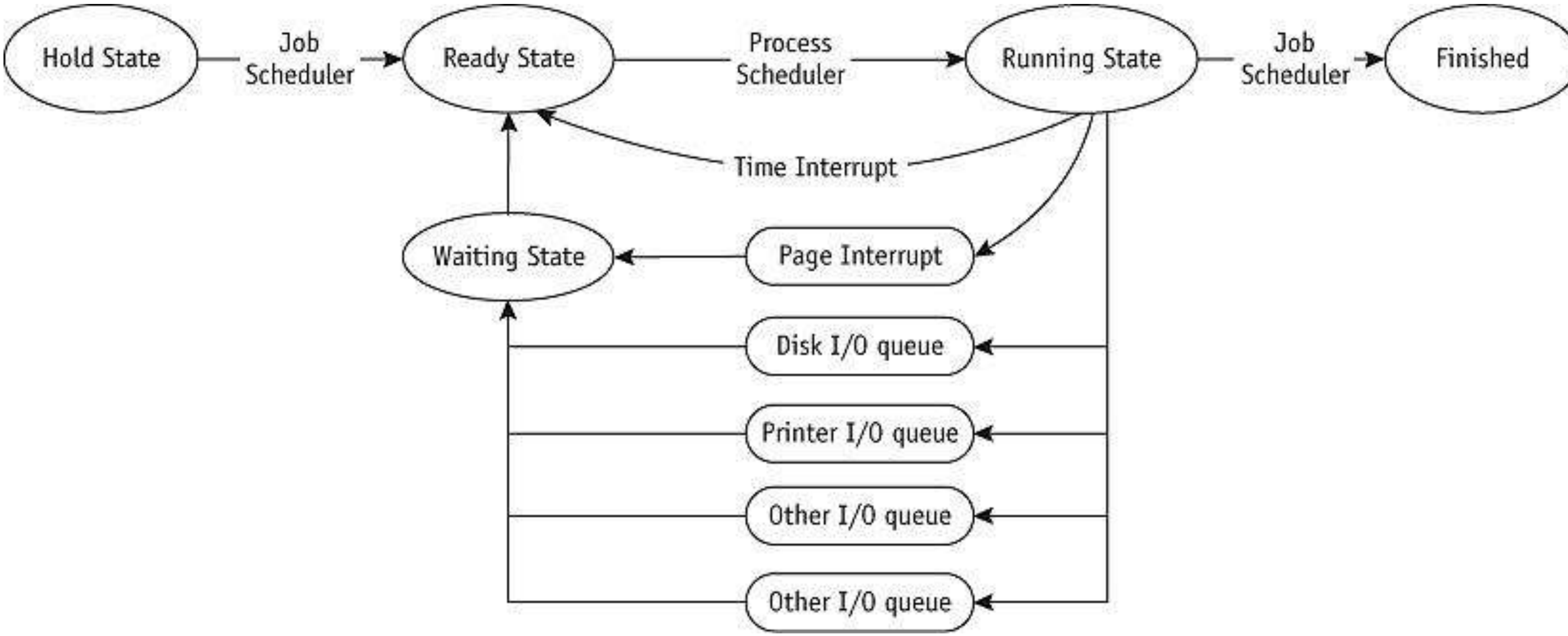
- ***Process id****. A unique number assigned when job enters the system*
- ***Process status****: hold, ready, running waiting*
- *Process state (all information to indicate current state of the job*
  - ***Program counter****: current instruction counter*
  - ***register contents****; e.g. accumulator register*
  - ***Main memory****: where the job is stored by refer to process management table (P.M.T.)*
  - ***Resources****: hardware (disk) or files*
  - ***Priority****: determines when the job will run*
- ***Accounting****: what resources, e.g. CPU, the job used and for how long*

Managing process List (URL double link list link)



Task_struct is linux implementation of the PCB

Processing states (Control Block) and Queuing (cont'd.)



(figure 4.5)
Queuing paths from HOLD to FINISHED. The Job and Processor schedulers release the resources when the job leaves the RUNNING state.

Scheduling Policies

- Multiprogramming environment
  - More jobs than resources at any given time
- Operating system pre-scheduling task
  - Resolve three system limitations
    - Finite number of resources (disk drives, printers, tape drives)
    - Some resources cannot be shared (mutual exclusion) once allocated (printers)
    - Some resources require operator intervention before reassigning: mouse click / keyboard…

Scheduling Policies (cont'd.)

- Good process scheduling policy criteria (cont'd.)
  - Maximize CPU efficiency
    - Keep CPU busy 100 percent of time
  - Ensure fairness for all jobs
    - Give every job equal CPU and I/O time

- Final policy criteria decision lies with system designer or administrator (**select a scheduling policy that maximises efficiency of system resources**)

Scheduling Policies (cont'd.)

- Problem
  - Job claims CPU for very long time before I/O request issued
    - Builds up READY queue and empties I/O queues
    - Creates unacceptable system imbalance
- Corrective measure
  - Interrupt
    - Used by Process Scheduler upon predetermined expiration of time slice
    - Current job activity suspended
    - Reschedules job into READY queue

Scheduling Policies (cont'd.)

- Types of scheduling policies (will be referred to in process scheduling lecture)
  - Preemptive
    - Used in time-sharing environments
    - Interrupts job processing after a specific amount of time if there has been no other interrupts
    - Transfers CPU to another job (round robin)
  - Non-preemptive
    - Functions without “external” interrupts
    - a job or process remains in a running state until interrupted by internal interrupts e.g. I/O request**:** **used in batch systems**

Re-scheduling a process

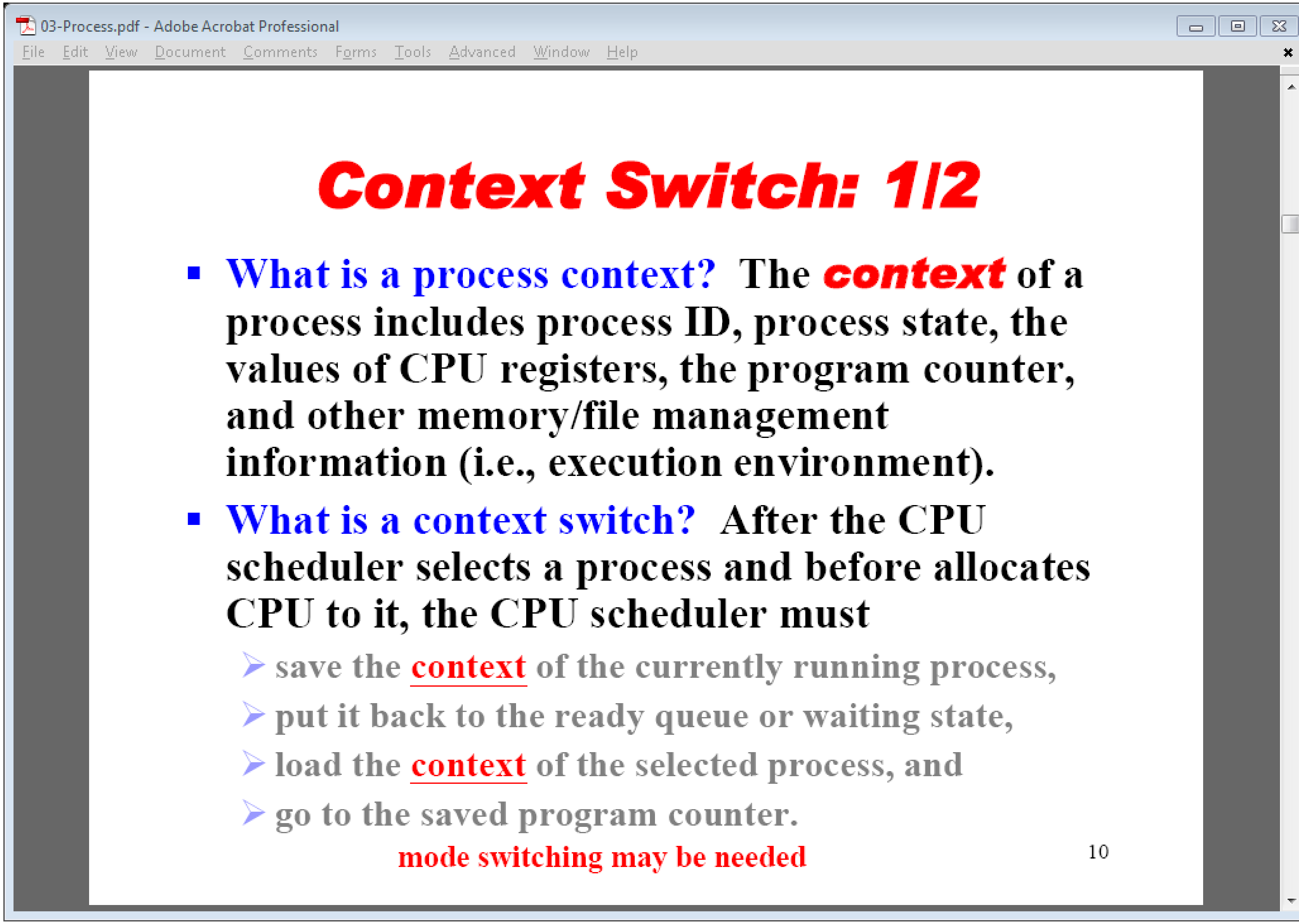
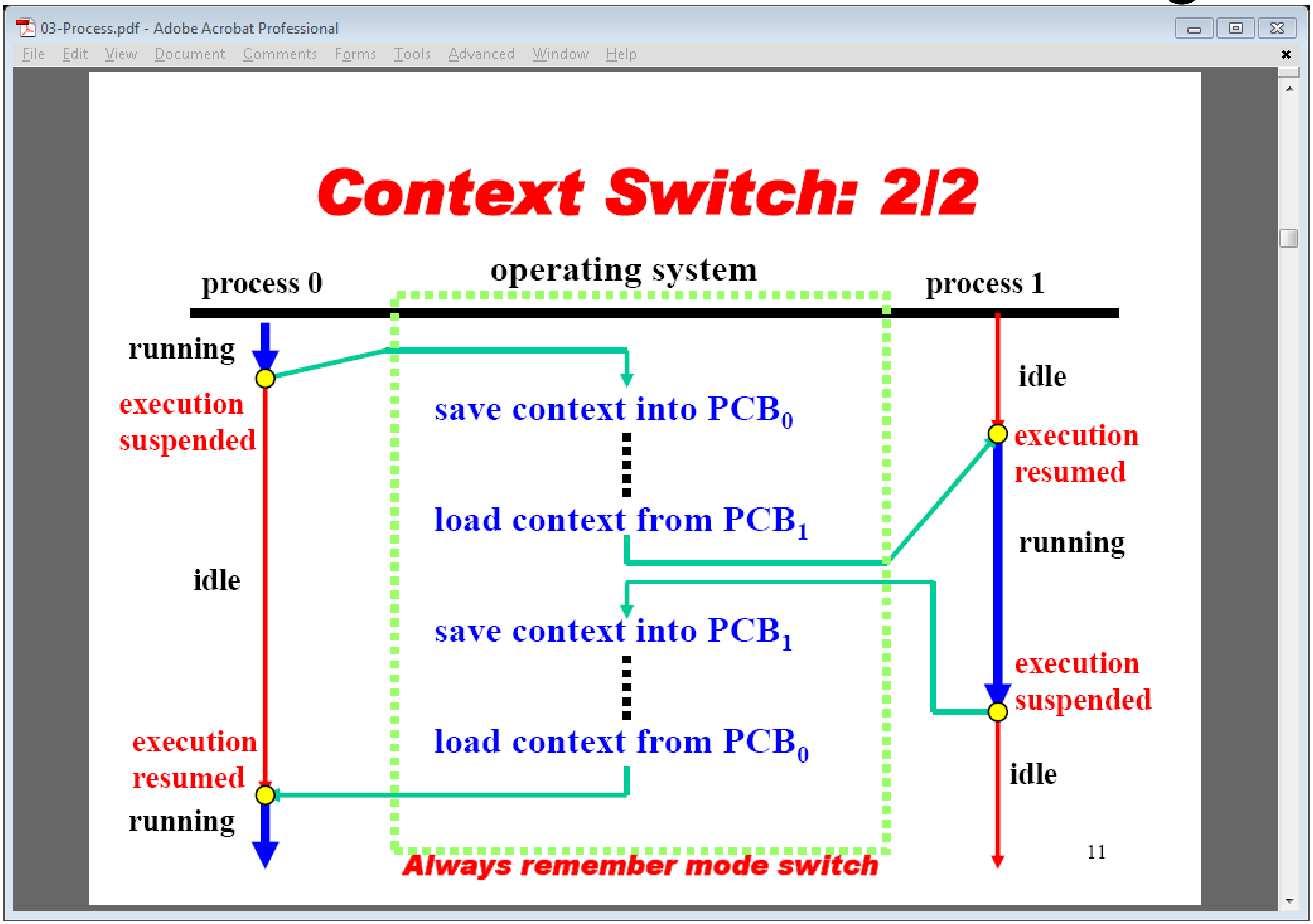


Illustration of context switching



Lists/Queues and content switching

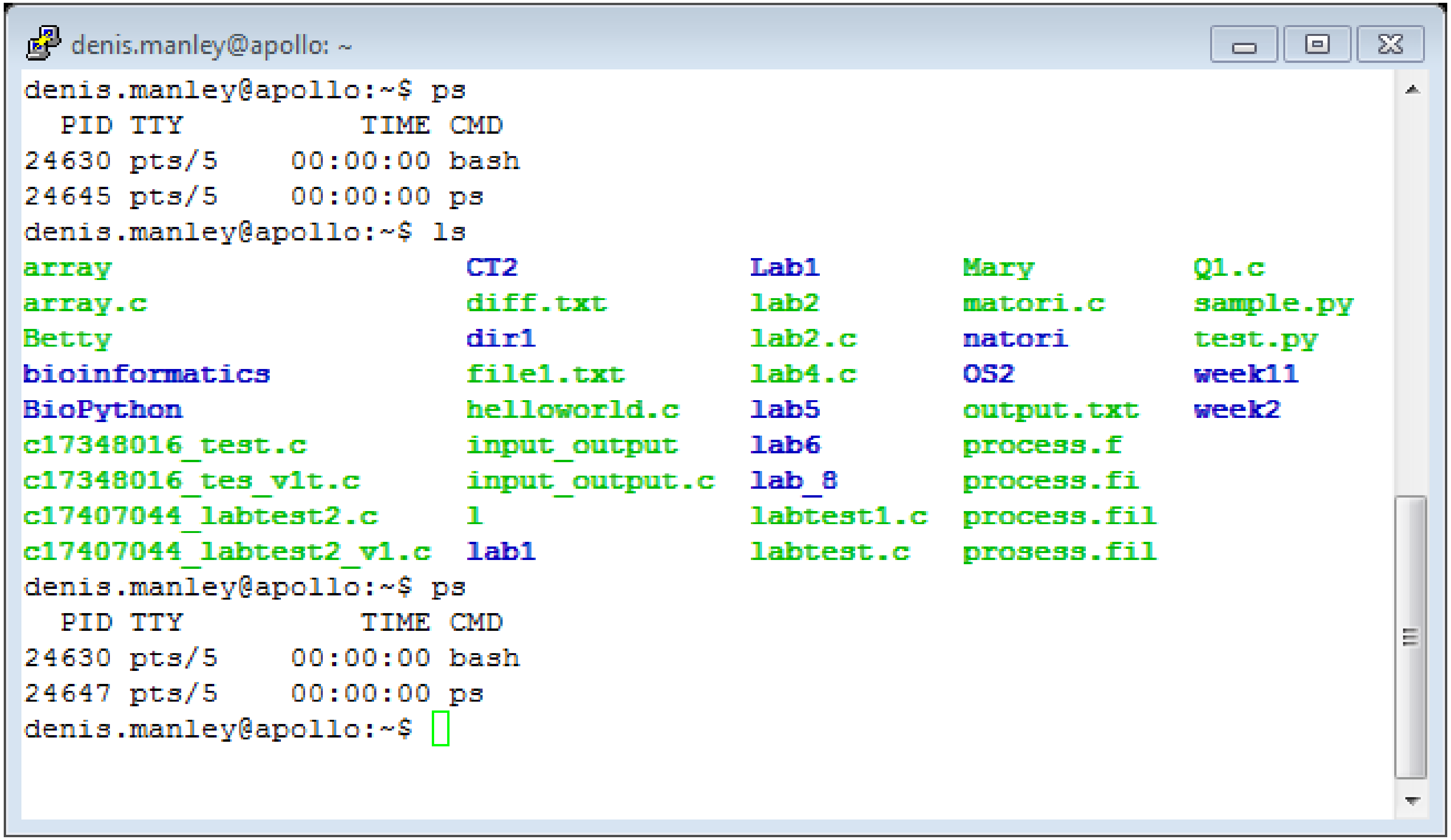
- Process management is a combination of different Queues (enqueue/dequeue/ headPtr/TailPtr for each…)
- Ready Queue: enqueue/dequeue: Node contains a process ID…
- On Context switching the PCB structure is update on the last (edit contents).
  - The process ID is dequeued from head ready queue.
  - Used to find corresponding PCB node (currentPtr moved to node)
    - PCB (context) Data written from node to registers
    - The call stack register is updated to process call stack
    - The program counter register is updated for new process
  - The other queues continue to enqueue as appropriate

- Scheduling polices that result in priority queues

Executing a program in linux:

- The following shots that the bash is an active process (note ps command show active processes)
- The ls command (program is shown at the prompt before execution: the bash is currently the only active process. running
- On pressing return 3 things are happening:
  - The bash command code is copied to a different location (different PID): the processes is forked
  - The parent process (bash) is forced to stop executing (Wait)
  - The ls code over writes the copied bash code and executes: this displays all files and directories
  - When it is finished the bash/prompt of parent process is reactivated (continues with execution)

Screen shot of running a process



Sample Questions

- Briefly describe the elements of a PCB (4 marks)
- Explain the steps a process can undergo as through the job and process scheduler. (8 marks)

- Describe the process of context switching as it applies to process control blocks (PCB). (5 marks)

- In linux a process is created using the fork() command. Explain the steps that occur if a fork() command is called (8 marks)
-

Sample Question

The wait () command is another important associated with processes: Explain, using a suitable example, exactly how the wait functions works (10 marks).

- The execvp command has the following format:
  - *int execvp(char *prog, char *argv[])*
- Explain what values are stored in the prog and the argv parameters if the two command line argument are: *gcc* and *file1.c* (4 marks)

Explain how the following code using a combination of the fork(), wait() and exec() command can be used to run new processes. (12 marks)

Sample Question

```
•   if ((pid = fork()) < 0) {     /* fork a child process            */
•           printf("*** ERROR: forking child process failed\n");
•           exit(42);
•       }
•       else if (pid == 0) {          /* for the child process:         */
•           if (execvp(*argv, argv) < 0) {     /* execute the command  */
•               printf("*** ERROR: exec failed\n");
•               exit(1);
•           }
•       }
•       else {                                  /* for the parent:      */
•           while (wait(&status) != pid)       /* wait for completion  */
•               ;
•       }
```