CMPT 300
Operating System I
Midterm Review

Dr. Hazra Imran

Admin notes

- Midterm date (June 20 from 1:30 -2:20 pm)
  - 50 min exam
  - In person exam only.
  - No cheat sheets. No sample midterm exam. I will provide some sample questions.

- Content: All lecture and material covered prior to June 17.

- Suggested way to Study
  - Start with the post-lecture slides (special attention to the list of Learning Goals.
  - Textbook (focus on the lecture notes)
  - Go through Quizzes, In-class activities, Clickers

Tentative Number of Problems (subject to change)

4-5 True or False

- Choosing T or F, no explanation needed or allowed
- Full credit or zero credit

4-5 Multiselect questions

Full credit or zero credit

3-4 Short questions.

- Write outcome of a code piece
- Short explanation require
- Partial credit

Midterm Topic

- Coverage
  - Chapter 1 : Introduction
  - Chapter 2 : OS Structure
  - Chapter 3: Process
  - Chapter 4: Threads
  - Chapter 5 : Scheduling

Midterm Review Suggestions

NOTE: This is not guaranteed to be exhaustive!

Chapter 1

- Organization of Computer System
- Role of interrupts
- User mode vs kernel mode
- What is Operating System

Interrupts/DMA

- Basic concept of interrupt
  - Signals sent by devices to the CPU when some event happens
- CPU runs much faster than other devices
- Using interrupts allows the CPU to do other tasks while an I/O is in progress
  - Steps of handling an interrupt ✓
- Two ways of doing I/O operations
  - Busy waiting – wastes CPU cycles
  - Interrupts – better than busy waiting, but also depends on the frequency CPU is interrupted
- DMA operations allow I/O devices to transfer data to/from memory without CPU intervention

Chapter 2

- Services provided by OS
- System Calls
- OS Structure

OS Structures

- Monolithic
- Layered
- Microkernel
- Modular
- Hybrid

Pros and Cons?

- Modern OSes are often hybrids ✓
    - Exhibit properties from multiple structures

Chapter 3

- Process Operations
- Process Sates
- Process Control Block
- Context switch
    - switch from one process to another
    - Need to save and restore process states
    - Pure overhead
    - fork() exec ( )
    - Signals

- Inter Process Communications (IPC) Mechanisms (Shared Memory, Message Passing , Pipe)

Chapter 4

- Thread
- User and Kernel Threads
- Pthreads

Processes and threads

- Similarities and differences?
- What’s shared between threads
- How do processes/threads communicate with each other?

Process and Threads

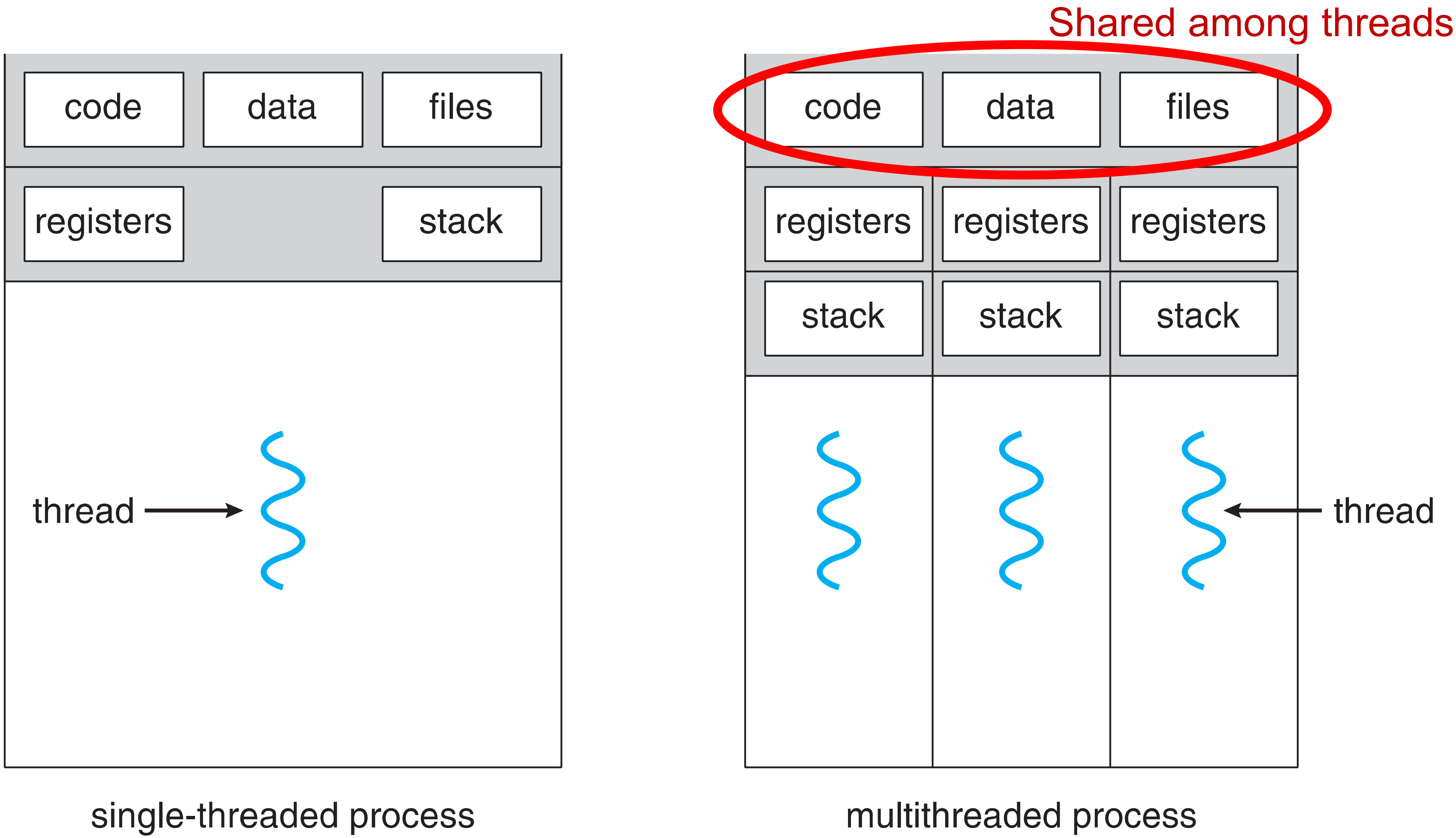
Process: program in execution

- Program on disk (file) is not a process, until it is loaded in memory and required OS structures are created
- Terms "job" and "process" are interchangeable

Threads: a basic unit of CPU utilization

- Threads in the same process share certain resources
  - What are they?
- But not everything is shared
  - What are the things that are not shared?

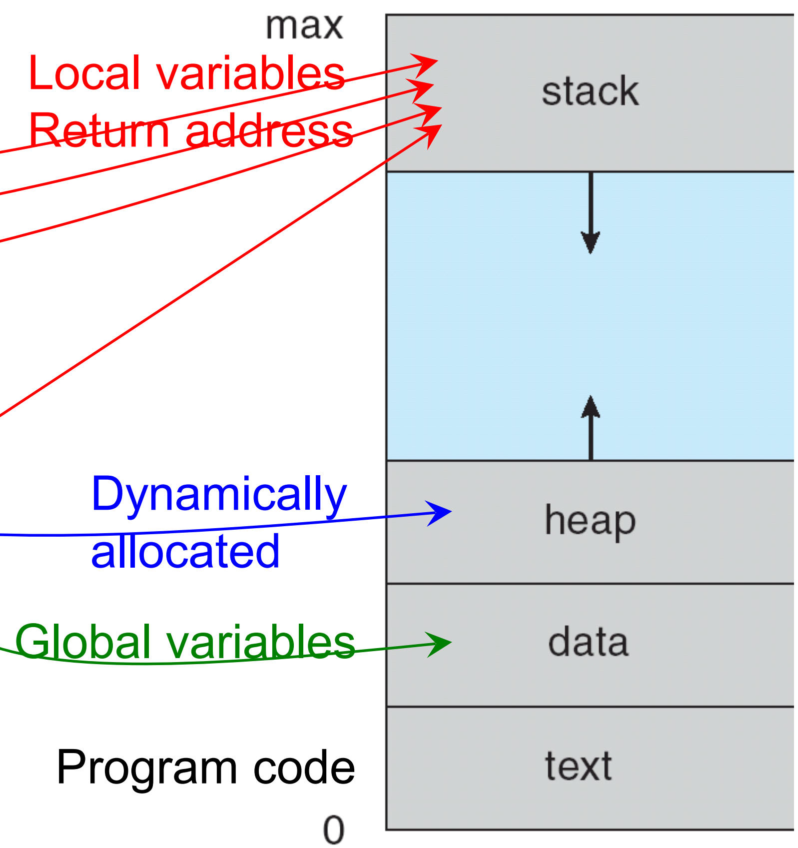
Single and Multithreaded Processes



Process in Memory

```
int global = 0;
int main (int arg)
{
      float  local;
      char *ptr;

      ptr = malloc(100);
      local = 0;
      local += 10*5;
      …..
      ….
      foo();
      …. /* return addr */
      ….
      return 0;
}
```

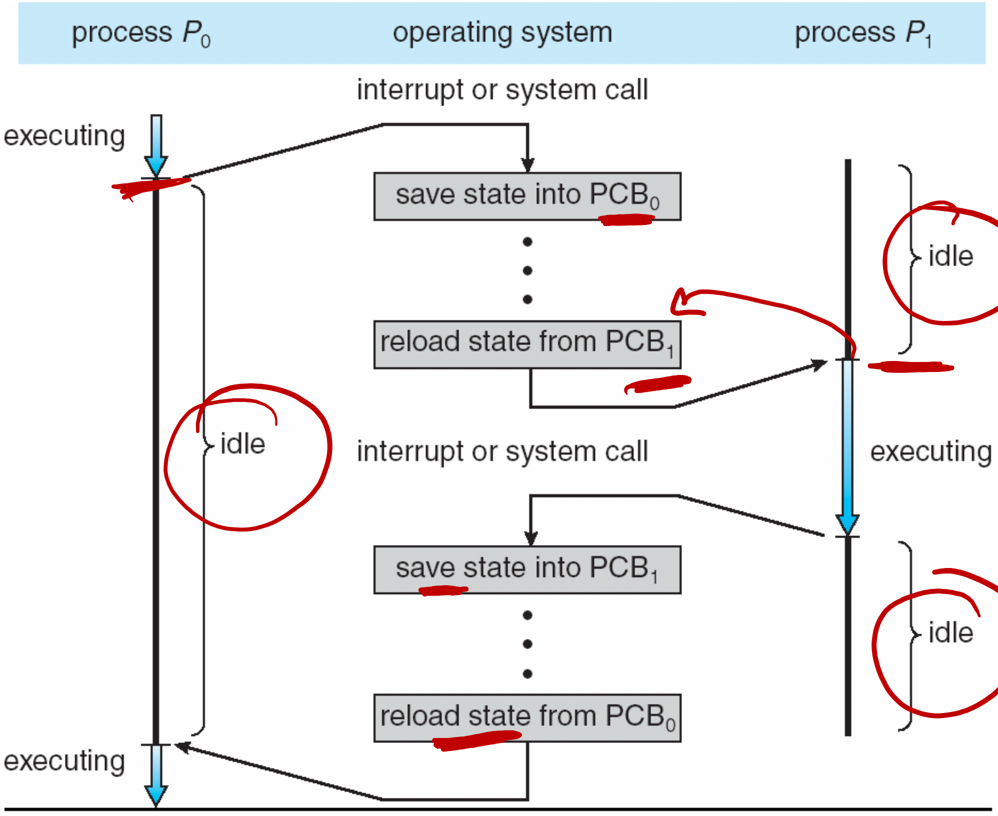


Processes: Context Switch

Switch the CPU core from one process to another

- When switching occurs, the OS kernel:
  - Saves state of P0 in PCB0 (in memory)
  - Loads state of P1 from PCB1 into registers

State: values of the CPU registers (program counter, stack pointer, etc.)



C Program Forking a Child Process

```
int main() {
    pid_t  var_pid;
    var_pid = fork();
    if (var_pid < 0) {
        fprintf (stderr, "fork Failed");
        exit(-1);
    }
    else if (var_pid == 0) { /* child process */
        execlp ("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for child to complete */
        wait (NULL);
        printf ("Child %d Completed", var_pid);
        exit(0);
    }
}
```

fork returns twice

exec does not return if succeeded

Note: fork returns 0 to child and the pid of the new process to parent.

Example: Pthreads

string to integer

```
#include <pthread.h>
int sum;
void* runner(void* param) {
    int i, upper = atoi(param); sum = 0;
    for (i = 1; i <= upper; i++)
        sum += i;
    pthread_exit(0);
}

int main(int argc, char* argv[]) {
    pthread_t tid;  pthread_attr_t attr;

    pthread_attr_init(&attr);  // default attributes
    pthread_create(&tid, &attr, runner, argv[1]);
    pthread_join(tid, NULL);

    printf("Sum = %d\n", sum);
    return 0;
}
```

Chapter 5

- Goals of scheduling (What to maximize? What to minimize?)
- Process Cycles (CPU Burst and I/O Burst)
- Process Queues (Ready and I/O)
- Context Switching
- Thread Scheduling
- Preemptive and Non-preemptive Scheduling
- First Come First Serve Scheduling
- Shortest Job First Scheduling
- Shortest Remaining Time First Scheduling
- Priority Scheduling
- Round Robin Scheduling
- Multiple Queues Scheduling and Multiple queue feedback Scheduling

Scheduling

- Process/thread types
  - I/O-bound: spend more time on I/O
  - CPU-bound: spend more time on CPU
  - Examples?
- Scheduling can be
  - Preemptive: OS scheduler can force a process to give up CPU
  - Non-preemptive: process only voluntarily leaves the CPU
- Algorithms
  - FCFS, SJF, Priority, Round-Robin, Queue based
  - Properties of each algorithm

Scheduling Metrics

- Maximize (higher = better)
    - CPU utilization – keep the CPU as busy as possible
    - Throughput – # of processes that complete their execution per time unit

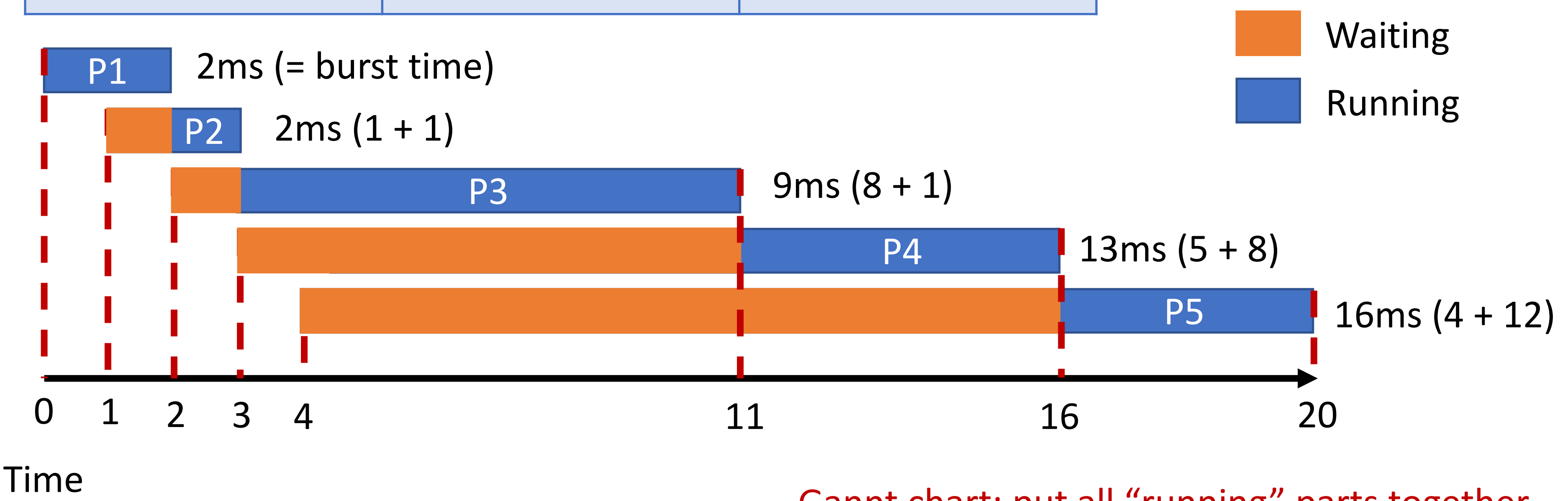
- Minimize (lower = better)
    - Turnaround time – amount of time to execute a particular process (time from submission to termination)
    - Waiting time – amount of time a process has been waiting in ready queue
    - Response time – amount of time it takes from when a request is submitted until the first response is produced

Scheduling Examples

Process	Burst Time (ms)	Arrival Time (AT)
P1	2	0
P2	1	1
P3	8	2
P4	5	3
P5	4	4

Turnaround time for each process under FCFS?

Burst time + wait time



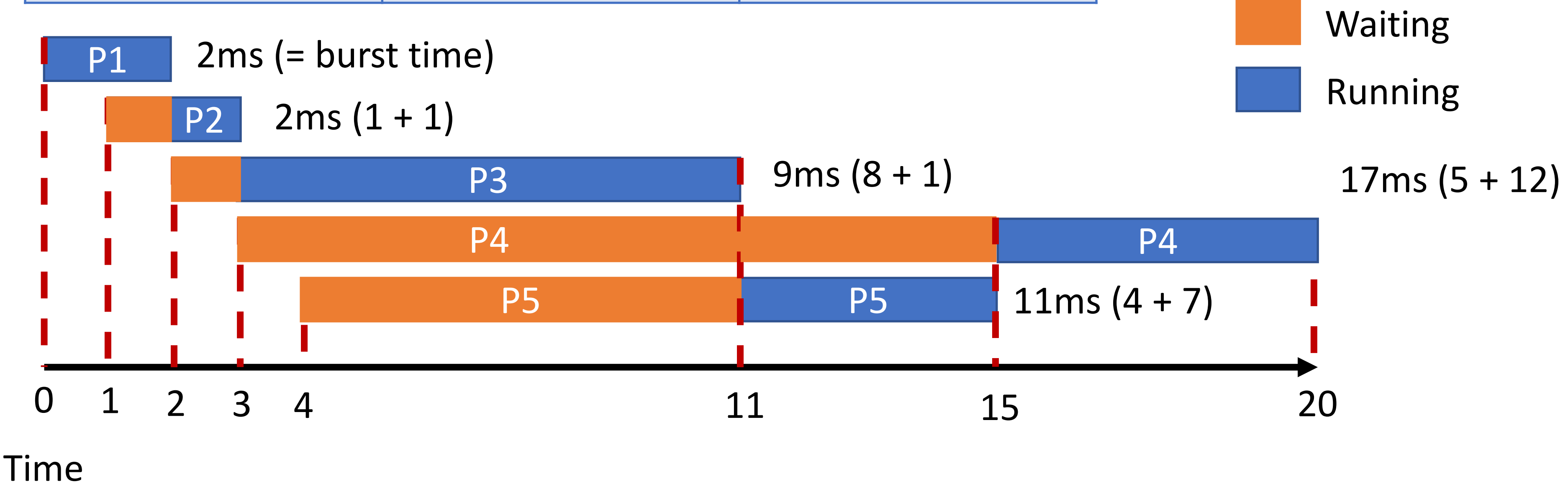
Gannt chart: put all "running" parts together

Scheduling Examples

Process	Burst Time (ms)	Arrival Time (AT)
P1	2	0
P2	1	1
P3	8	2
P4	5	3
P5	4	4

Turnaround time for each process under Shortest-Job-First?

Burst time + wait time



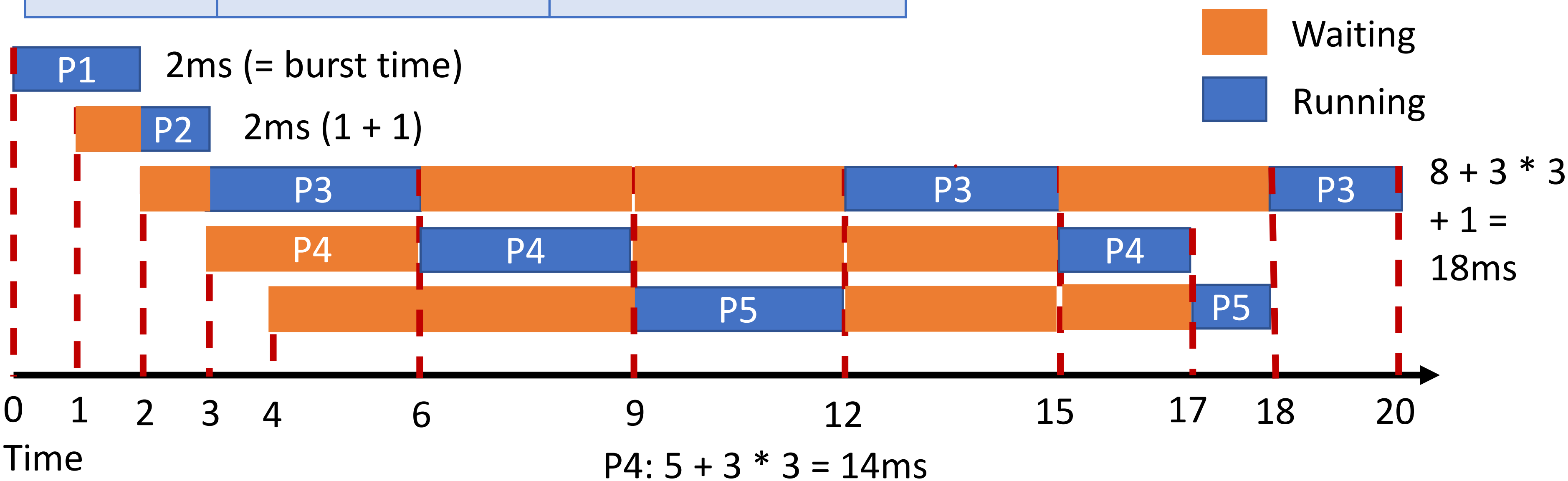
Scheduling Examples

Process	Burst Time (ms)	Arrival Time (AT)
P1	2	0
P2	1	1
P3	8	2
P4	5	3
P5	4	4

Turnaround time for each process under Round-Robin?

Assume: time quantum (q) = **3ms**

Burst time + wait time



P4: 5 + 3 * 3 = 14ms

P5: 4 + (2 + 3 * 2 + 2) = 14ms

Sample questions

- What are the roles of the OS?
- What does it mean to share the resources of the system?
- What is a context switch?
- Define Interrupts.
- Define preemption.
- What is a process?
- What is the difference between a process and a program?
- What is contained in a process?
- What is PID?
- What information does PCB contain?
- How is it used in a context switch?
- What different states can a process be in?
- When does a process change state?
- What does fork()?
- What does it return?
- What does exec() do?

Sample questions

- How is exec() different from the fork?
- When you type ls -l at a bash prompt, how do ls get the -l argument?
- What are system calls, and how are they handled?
- What are interrupts, and how are they handled?
- What is a thread?
- What is the difference between a thread and a process?
- How are they related?
- Why are threads useful?
- Why does each thread have its own stack?
- How are threads managed by the system?
- What is a thread control block? TCB
- User-level and kernel-level threads
  - What's the difference?
  - What are the advantages/disadvantages of one over another?
  - Different user-level kernel-level thread mapping models

Sample questions

Consider the following code snippet

```c
int value = 10;

    int main ( ) {
        //...
    if ( fork() == 0 )
        value += 15;
    else {
        wait(NULL);
        printf ("\n value = %d \n", value);    /*Line A */
    }
// …
}
```

(a) What will be the output of Line A?
(b) If we remove the statement wait(NULL), what will be the output?

Sample questions

Consider the following set of processes:

Process	Arrival Time	Burst Time	Priority
P1	0	10	3
P2	1	1	1
P3	1	2	2
P4	2	1	4
P5	4	5	1

Consider a preemptive priority CPU scheduling policy. Note that a smaller number implies higher priority.

(a) Draw a Gantt chart showing the execution of these processes.

(b) Compute the average waiting time and average turnaround time.

Finally, DON'T PANIC!

The best advice I can give about what to do during an exam is “**DON’T PANIC”**`.

When you panic, your mind freezes up, everything seems harder (even the easy stuff), and you get nothing done, which is the worst-case scenario in an exam setting.

Unfortunately, we all panic from time to time. It will happen.

So, it will be very useful to practice not panicking.

Some Tips

- Take some time to relax, calm down, and give yourself a silent pep-talk.
  - You've prepared for this exam.
  - You've already done some of it (or practice questions, clickers like it), and you can do more.
  - Part marks are your friend.
  - Even the worst-case scenario is not unrecoverable; **EVERYTHING WILL BE ALRIGHT.**