STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

CS 492: Operating Systems

Scheduling

Instructor: Iraklis Tsekourakis

Email: itsekour@stevens.edu

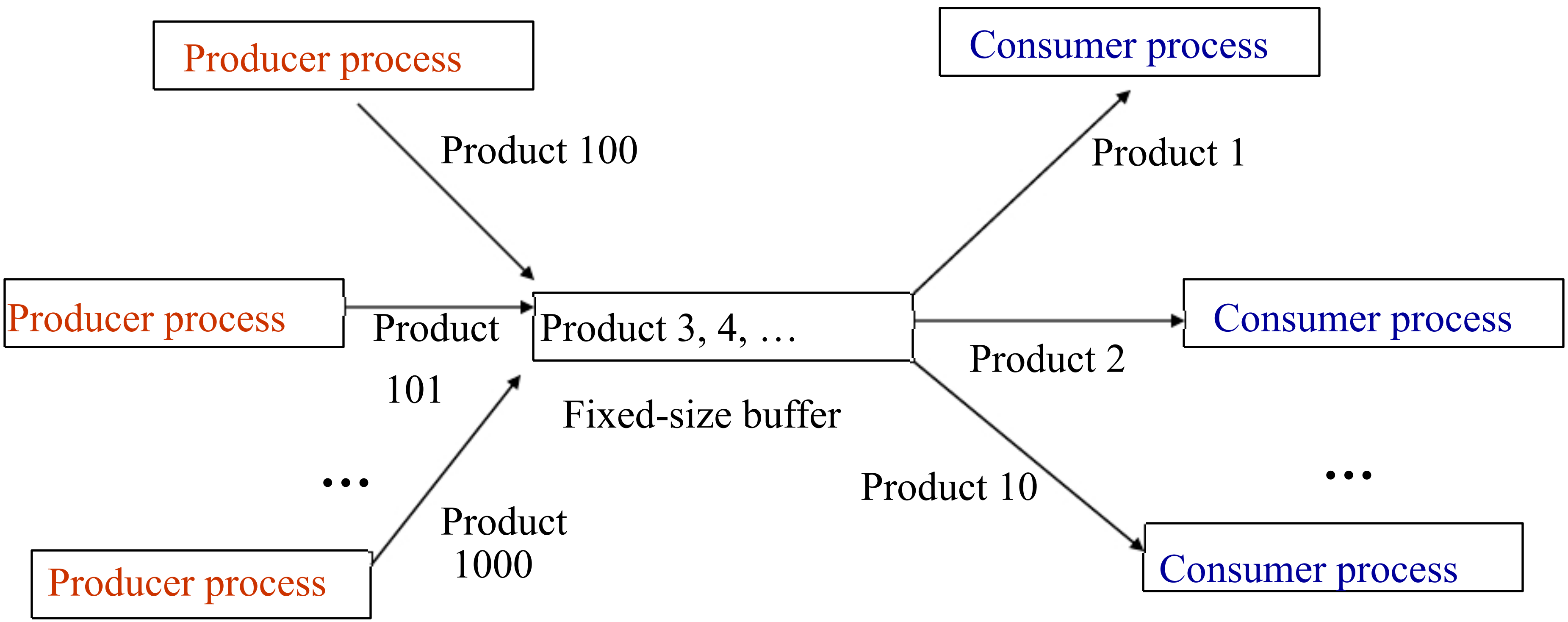
MIDTERM!

- Friday, March 9th
  - Computer Test on Canvas
  - Install lockdown browser

Assignment 1

(Due: 11:59pm, Monday, March 5th)

Assignment 1



Input Parameters

- P1: Number of producer threads
- P2: Number of consumer threads
- P3: Total number of products to be generated by all producer threads
- P4: Size of the queue to store products for both producer and consumer threads (0 for unlimited queue size)
- P5: 0 or 1 for type of scheduling algorithm
  - 0: First-Come-First-Serve;
  - 1: Round-Robin
- P6: Value of quantum used for round-robin scheduling
- P7: Seed for random number generator

Product

- Each product has:
  - A unique product ID;
  - The timestamp when the product is generated
  - The “life” of the product
- After a product is consumed, the consumer thread prints out the product ID, and sleeps 100 milliseconds (by calling usleep()).

Critical Section

- Protect the queue
  - When the queue is full
  - When the queue is empty
- Protect the shared variables (e.g., total number of products generated by all producer threads)

Scheduling (1/2)

- Producers: first come, first serve
- Consumer threads: two scheduling algorithms
  1. First come, first serve
     - Products are sorted in the order that they are inserted into the queue
     - The **consumption** of a product is simulated by calling the *fn(10)* function *l* times, where *l* is the product's life

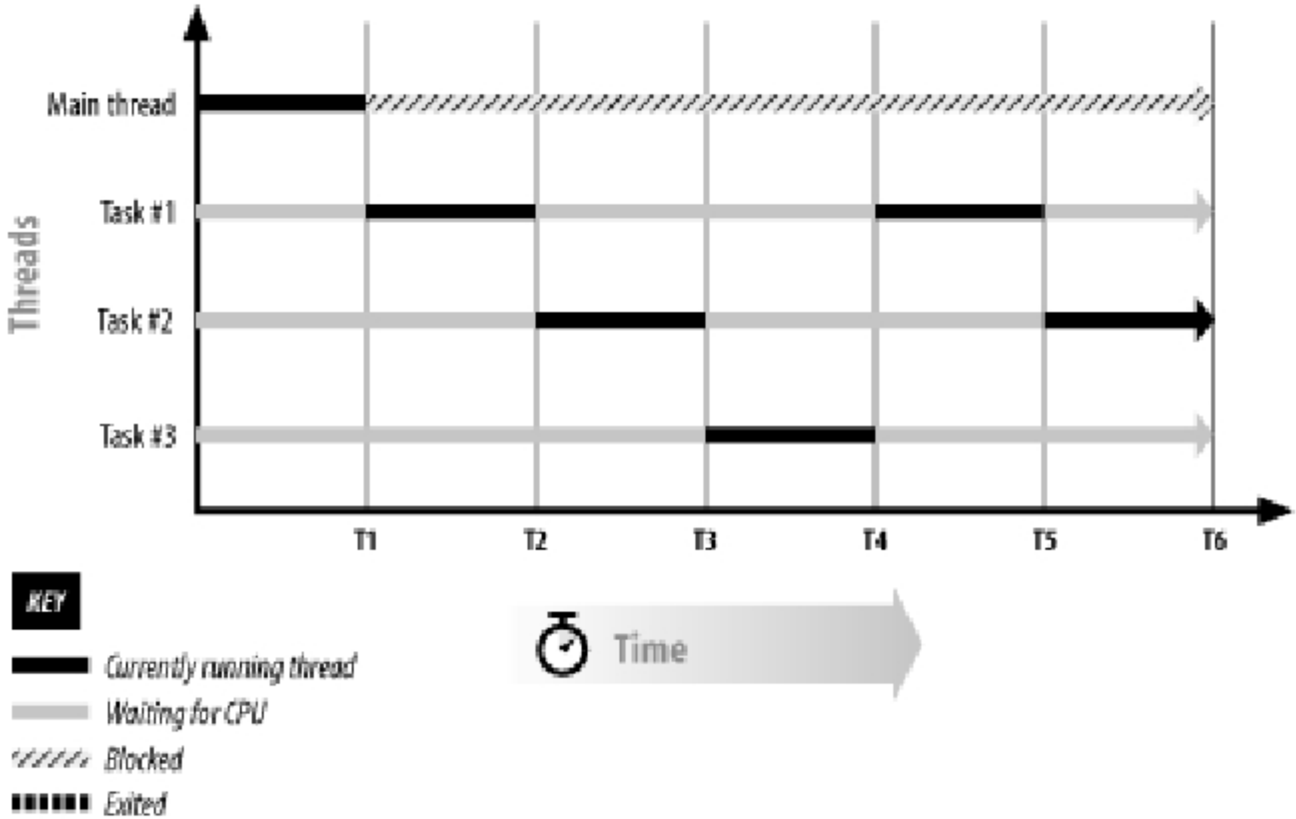
Scheduling (1/2)

- Producers: first come, first serve
- Consumer threads: two scheduling algorithms
  - First come, first serve
  - Round-robin
    - Each product is assigned a quantum *q*
    - If the product's life l>q
      - Call *fn(10)* function *q* times
      - Update product's life l = l – q, and keep product in the queue.
    - Otherwise,
      - Call *fn(10)* function *l* times
      - Print product ID, and remove product from queue.

Experiments and Reports

- Experiments: change the setting of the following parameters
  - Number of producers/consumers
  - Value of quantum used for round-robin algorithm
- Report: collect statistics of
  - Total time for processing all products
  - Min, max, and average *turn-around* times (the time between the product is produced and it is removed from queue)
  - Min, max, and average *wait* times (the time that products wait in the queue)
  - Producer throughput (number of products inserted into queue per minute)
  - Consumer throughput (number of products printed out per minute)
- More details/requirements are in the assignment description

Scheduling



Main thread
Task #1
Task #2
Task #3
Threads
T1
T2
T3
T4
T5
T6
KEY
Currently running thread
Waiting for CPU
Blocked
Exited
Time

Recap

- Inter-process communication (IPC)
  - Mechanisms for shared memory
    - Software-based techniques
    - Hardware-based techniques
    - Higher level (Semaphores, Mutex, Monitors)

What's Next?

- Multiple processes/threads are ready to run
- We assume single CPU scenario
  - So only one process/thread can be picked
- Which one?
- But, we have to worry about efficient use of CPU as well

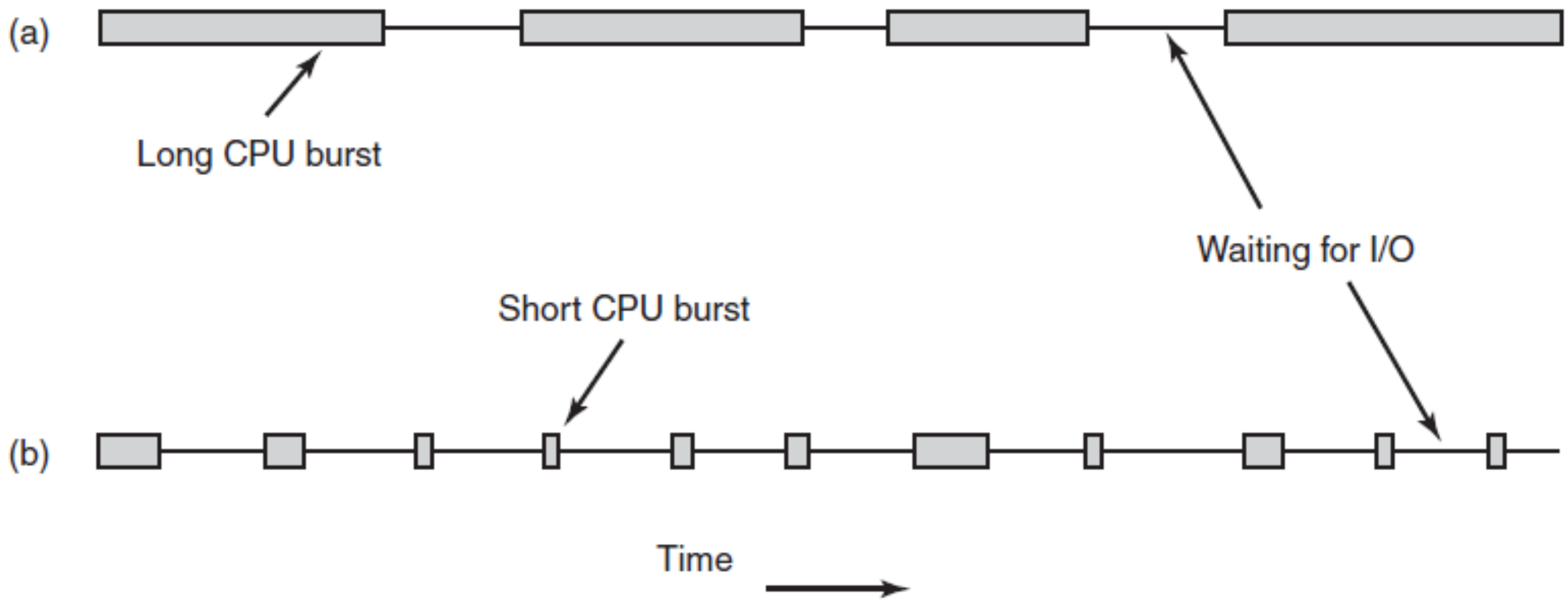
Goals for Today

- Scheduling
  - Issues in scheduling
  - Basic scheduling algorithms
    - Batch system
    - Interactive system
    - Real-time system

Scheduling

- The Problem
  - $K>1$ processes/threads that are ready
  - $N \geq 1$ CPUs (N may be $< K$)
  - Which processes/threads to assign to which CPU(s)?

Introduction to Scheduling Behavior

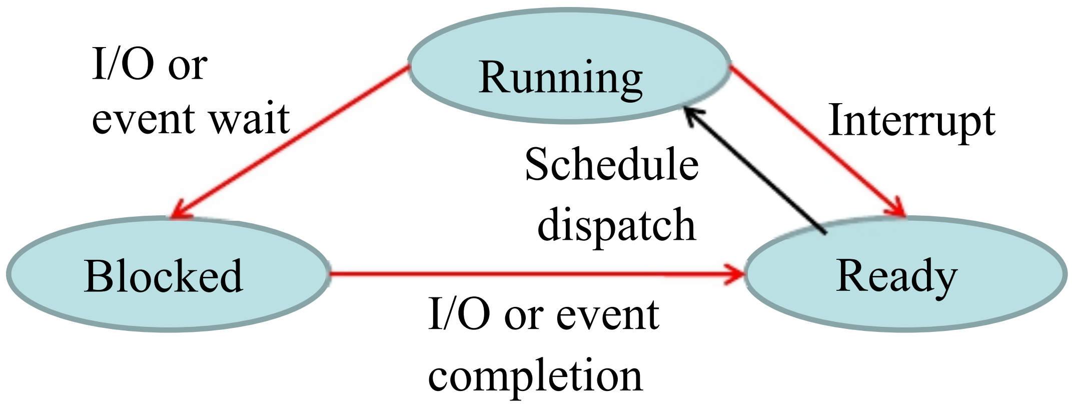


CPU-bound vs I/O-bound

Question?

- Can a measure of whether a process is likely to be CPU bound or I/O bound be determined by analyzing source code? How can this be determined at run time?

When to Schedule?



- Scheduling decisions may take place when

1. A process is created
2. A process in the running state exits
3. A process switches from *Running* to *Blocked* state (e.g., by semaphore)
4. A process switches from *Running* to *Ready* state
5. A process switches from *Blocked* to *Ready*

Clock Interrupts

- Scheduling takes place at clock interrupts
- Clock interrupts: how system keeps track of time
  - Interrupt at periodic intervals (clock ***tick***)
  - Typically 1msec
  - Implemented using a hardware clock interrupt
  - High priority, second to power-failure interrupt.

*However, Ivan Sutherland, http://www.cs.virginia.edu/~robins/Computing_Without_Clocks.pdf

Non-preemptive vs. Preemptive Scheduling

- *Non-preemptive* scheduling
  - For every clock interrupt, running process keeps going
- *Preemptive* scheduling
  - For every clock interrupt, running process is suspended and switched with another process (if there is any)
  - Maximum or fixed time execution of processes (timeslice)

Scheduling Criteria (1/2)

- User-oriented
  - Minimize *response* time - time from issuing request to get the first response
    - Time to echo a keystroke in editor
    - Time to compile a program
    - Real-time Tasks: Must meet deadlines
  - Minimize *turnaround* time – time between submission and termination
  - Maximize *throughput* – # of jobs completed per unit time
  - Freedom from starvation
  - Fairness

Scheduling Criteria (2/2)

- System-oriented
  - Maximize CPU *utilization* - keep the CPU as busy as possible
  - Minimize *overhead* – keep CPU doing useful work as opposed to context switching

Goals for Today

- Scheduling
  - Issues in scheduling
  - Basic scheduling algorithms
    - Batch system
    - Interactive system
    - Real-time

Scheduling in Batch Systems

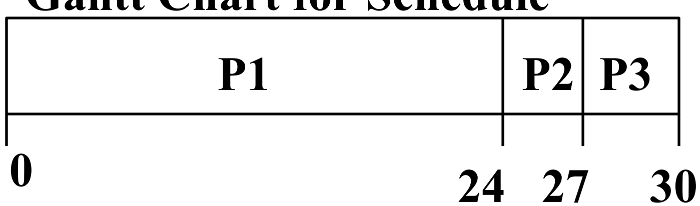
- Batch systems - off-line, jobs come with resource estimates
- Assumption: know how much time is needed to complete job ahead of time
- Scheduling goals
  1. Throughput: maximize jobs per unit time (hour)
  2. Turnaround time: minimize time users wait for jobs
  3. CPU utilization: keep the CPU as busy as possible
- Algorithms:
  – First-Come First-Served (FCFS)
  – Shortest job first (SJF)

First-Come, First-Served (FCFS) Scheduling

- Example

Process	CPUTime
P1	24
P2	3
P3	3

Gantt Chart for Schedule



turnaround time – average time takes for job to be completed after submission

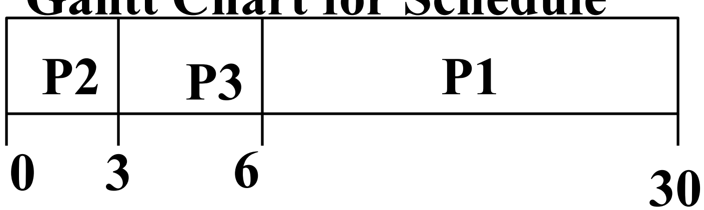
- Suppose the arrival order for the processes is P1, P2, P3
- Turnaround time
  - P1 = 24;
  - P2 = 27;
  - P3 = 30;
- Average turnaround time (24+27+30)/3 = 27
- Short process delayed by long process: *Convoy effect*

First-Come, First-Served (FCFS) Scheduling (Cont.)

- Example

Process	CPUTime
P1	24
P2	3
P3	3

Gantt Chart for Schedule



- Suppose now the arrival order for the processes changes: P2, P3, P1
- Turnaround time
  - P1 = 30
  - P2 = 3;
  - P3 = 6;
- Average turnaround time
  - (30+3+6)/3 = 13
- Much better than the previous case

Shortest-Job-First (SJR) Scheduling

- Associate with each process the length of its CPU time.
- Use the CPU time length to schedule the process with the shortest CPU time.
- Two variations:
  - *Non-preemptive* – once CPU given to the process it cannot be taken away until it completes.
  - *preemptive* – if a new process arrives with CPU time less than the remaining time of current executing process, preempt. (*Shortest Remaining Time Next)*

Non-Preemptive SJF Example

- Example

Process	ArrivalTi me	CPUTime
P1	0	7
P2	2	4
P3	4	1
P4	5	4

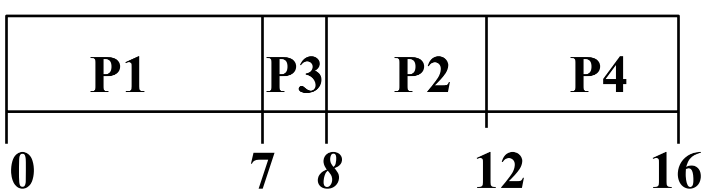
What's the turnaround time for each process?
What's the average turnaround?

Non-Preemptive SJF Solution

- Example

Process	ArrivalTime	CPUTime
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for Schedule



- Turnaround time
  - P1 = 7;
  - P2 = 12 – 2 =10;
  - P3 = 8 – 4 =4;
  - P4 = 16 – 5 =11
- Average turnaround time

(7+10+4+11)/4 = 8

Scheduling in Batch Systems: More Discussions

- Preemptive SJF is optimal – gives minimum average turnaround time for a given set of processes.
- Can use non-preemptive algorithms since this reduces process switching

Goals for Today

- Scheduling
  - Issues in scheduling
  - Basic scheduling algorithms
    - Batch system
    - Interactive system
    - Reatl-time system

Scheduling in Interactive Systems

- Scheduling goals
  - Response time: respond quickly to users' requests
  - Proportionality: meet users' expectations

- Algorithms:
  - Round-Robin Scheduling
  - Priority Scheduling
  - Multiple Queues

Round-Robin Scheduling

- Round-robin
  - Each process is allowed to run specified time quantum.
  - After this time has elapsed, the process is preempted and added to the end of the ready queue.

- Quantum – a time interval, e.g., 100 msec

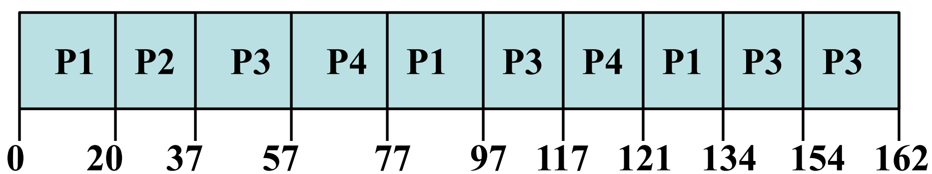
- Advantages:
  - Solution to fairness and starvation
  - Fair allocation of CPU across jobs
  - Low average waiting time when job lengths vary
  - Good for responsiveness if small number of jobs

Round Robin Example

- Time Quantum = 20

Process	CPUTime			
P1	53	**33 left**	**13 left**	
P2	17			
P3	68	**48 left**	**28 left**	**8 left**
P4	24	**4 left**		

Gantt Chart for Schedule



Typically, higher average turnaround time than SJF, but better response time