LECTURE 17

INTER-PROCESS COMMUNICATION AND PROCESS SCHEDULING

SUBJECTS

L'Université canadienne
Canada's university

Unicast vs Multicast

Message Passing (Synchronous vs Asynchronous)

CPU scheduling

- First Come First Served
- Round Robin

IPC – UNICAST AND MULTICAST

Distributed computing involves two or more processes engaging in IPC

- They use a pre-agreed upon protocol

A process may act as a sender at some point and a receiver at another

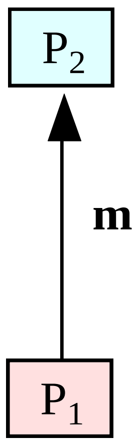
Unicast: communication between two processes

- E.g., Socket communication

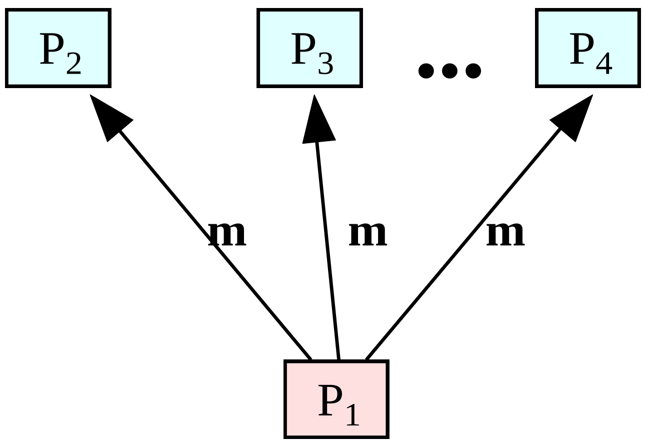
Multicast: communication between one process and a group of processes

- E.g., Publish/Subscribe Message model

UNICAST VS. MULTICAST



unicast



multicast

MESSAGE PASSING

uOttawa
L'Université canadienne
Canada's university

Message passing means that one process sends a message to another process and then continues its local processing

- The message may take some time to get to the other process

The message may be stored in the input queue of the destination process

- If the latter is not immediately ready to receive the message

Two types of message passing:

- Asynchronous message passing
- Synchronous message passing

ASYNCHRONOUS MESSAGE PASSING

Blocking send and receive operations:

- A receiver will be blocked if it arrives at the point where it may receive messages and no message is waiting.
- A sender may get blocked if there is no room in the message queue between the sender and the receiver
    - However, in many cases, one assumes arbitrary long queues, which means that the sender will almost never be blocked

ASYNCHRONOUS MESSAGE PASSING

Non-blocking send and receive operations:

- Send and receive operations always return immediately
    - They return a status value which could indicate that no message has arrived at the receiver
- The receiver may test whether a message is waiting and possibly do some other processing
    - It may optionally be notified by the system when a message is received

SYNCHRONOUS MESSAGE PASSING

One assumes that sending and receiving takes place at the same time

- There is often no need for an intermediate buffer

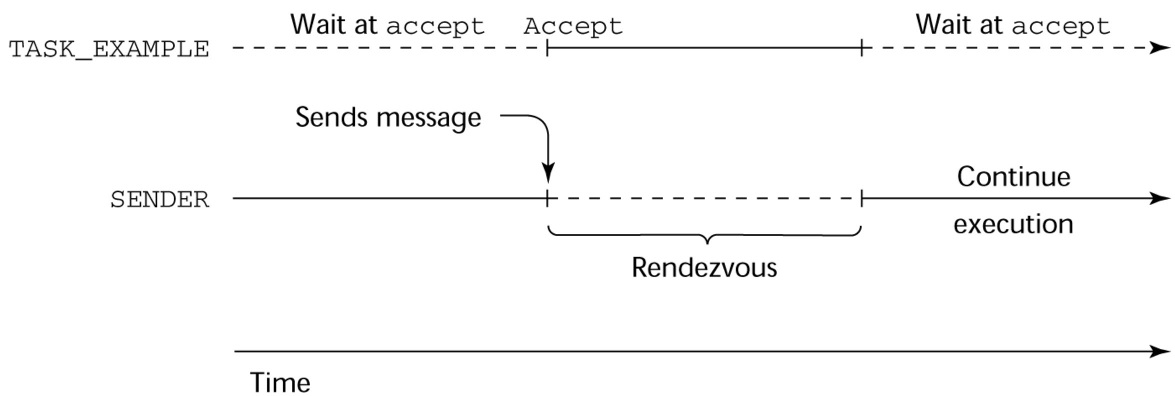
This is also called rendezvous and implies closer synchronization:

- The combined send-and-receive operation can only occur if both parties are ready to do their part.

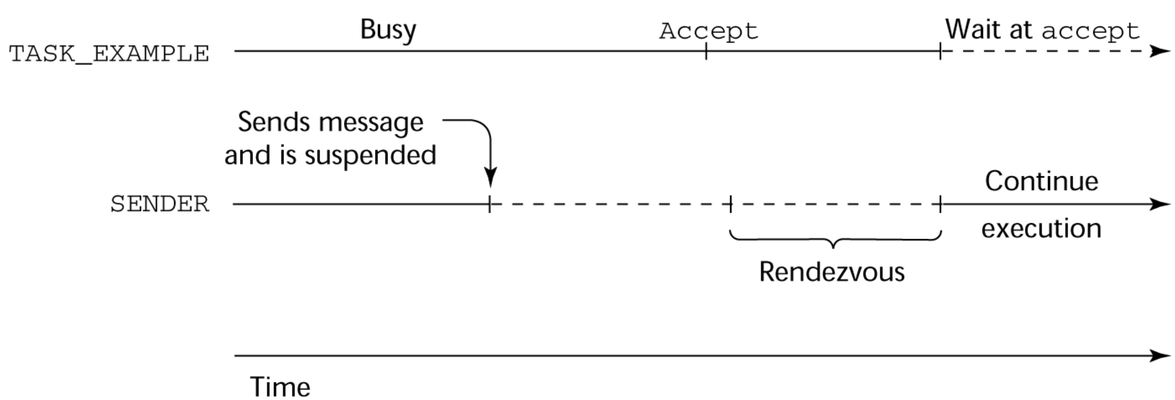
The sending process may have to wait for the receiving process, or the receiving process may have to wait for the sending one

RENDEZVOUS

(NOT AS ROMANTIC AS IT SOUNDS!)



(a) TASK_EXAMPLE waits for SENDER



(b) SENDER waits for TASK_EXAMPLE

CPU SCHEDULING

How does the OS decide which of several processes to take off the ready queue?

Scheduling: deciding which processes are given access to resources from moment to moment.

ASSUMPTION: CPU BURSTS

uOttawa
L'Université canadienne
Canada's university

Execution model: processes alternate between bursts of CPU and I/O

- Process typically uses the CPU for some period of time, then does I/O, then uses CPU again
- Each scheduling decision is about which process to give to the CPU for use by its next CPU burst
- With time slicing, process may be forced to give up CPU before finishing current CPU burst.

WHAT IS IMPORTANT IN A SCHEDULING ALGORITHM?

WHAT IS IMPORTANT IN A SCHEDULING ALGORITHM?

Minimize Response Time

- Elapsed time to do an operation (job)
- Response time is what the user sees
  - Time to echo keystroke in editor
  - Time to compile a program
  - Real-time Tasks: Must meet deadlines imposed by World

WHAT IS IMPORTANT IN A SCHEDULING ALGORITHM?

Maximize Throughput

- Jobs per second
- Throughput related to response time, but not identical
    - Minimizing response time will lead to more context switching than if you maximized only throughput
- Minimize overhead (context switch time) as well as efficient use of resources (CPU, disk, memory, etc.)

WHAT IS IMPORTANT IN A SCHEDULING ALGORITHM?

Fairness

- Share CPU among users or processes in some equitable way
- Not just minimizing average response time

SCHEDULING ALGORITHMS: FIRST-COME, FIRST-SERVED (FCFS)

L'Université canadienne
Canada's university

"Run until Done:" FIFO algorithm

In the beginning, this meant one process runs non-preemtively until it is finished

- Including any blocking for I/O operations

Now, FCFS means that a process keeps the CPU until it completes its burst

Example: Three processes arrive in order P1, P2, P3.

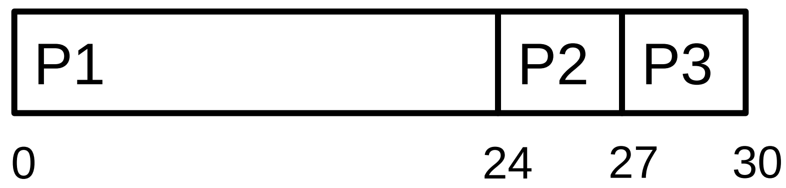
- P1 burst time: 24
- P2 burst time: 3
- P3 burst time: 3

Draw the Gantt Chart and compute Average Waiting Time and Average Completion Time.

SCHEDULING ALGORITHMS: FIRST-COME, FIRST-SERVED (FCFS)

Example: Three processes arrive in order P1, P2, P3.

- P1 burst time: 24
- P2 burst time: 3
- P3 burst time: 3



Waiting Time

- P1: 0
- P2: 24
- P3: 27

Completion Time:

- P1: 24
- P2: 27
- P3: 30

Average Waiting Time: (0+24+27)/3 = 17

Average Completion Time: (24+27+30)/3 = 27

SCHEDULING ALGORITHMS: FIRST-COME, FIRST-SERVED (FCFS)

uOttawa
L'Université canadienne
Canada's university

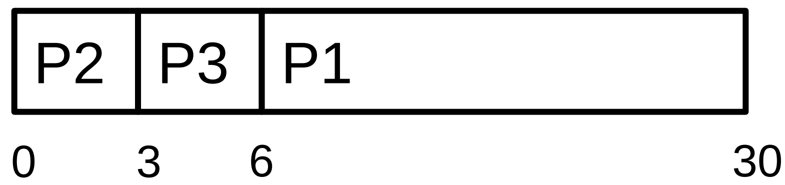
What if their order had been P2, P3, P1?

- P1 burst time: 24
- P2 burst time: 3
- P3 burst time: 3

SCHEDULING ALGORITHMS: FIRST-COME, FIRST-SERVED (FCFS)

What if their order had been P2, P3, P1?

- P1 burst time: 24
- P2 burst time: 3
- P3 burst time: 3



Waiting Time

- P2: 0
- P3: 3
- P1: 6

Completion Time:

- P2: 3
- P3: 6
- P1: 30

Average Waiting Time: (0+3+6)/3 = 3 (compared to 17)

Average Completion Time: (3+6+30)/3 = 13 (compared to 27)

SCHEDULING ALGORITHMS: FIRST-COME, FIRST-SERVED (FCFS)

Average Waiting Time: (0+3+6)/3 = 3 (compared to 17)

Average Completion Time: (3+6+30)/3 = 13 (compared to 27)

FCFS Pros and Cons:

- Simple (+)
- Short jobs get stuck behind long ones (-)
- Performance is highly dependent on the order in which jobs arrive (-)

HOW CAN WE IMPROVE ON THIS?

ROUND ROBIN (RR) SCHEDULING

uOttawa
L'Université canadienne
Canada's university

Round Robin Scheme

- Each process gets a small unit of CPU time (time quantum)
  - Usually 10-100 ms
- After quantum expires, the process is preempted and added to the end of the ready queue

- Suppose N processes in ready queue and time quantum is Q ms:
  - Each process gets 1/N of the CPU time
  - In chunks of at most Q ms
  - What is the maximum wait time for each process?
    - ***No process waits more than (N-1)Q time units***

ROUND ROBIN (RR) SCHEDULING

Performance Depends on the value of Q

- Small Q => interleaved
- Large Q is like **FCFS**
- Q must be large with respect to **context switch time**, otherwise overhead is too high
  - Spending most of your time context switching!

EXAMPLE OF RR WITH TIME QUANTUM = 4

<u>Process</u>	<u>Burst Time</u>
P_1	24
P_2	3
P_3	3

The Gantt chart is:

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

EXAMPLE OF RR WITH TIME QUANTUM = 4

Process	**Burst Time**
P_1	**24**
P_2	**3**
P_3	**3**

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

Waiting Time:

- P1: (10-4) = 6
- P2: (4-0) = 4
- P3: (7-0) = 7

Completion Time:

- P1: 30
- P2: 7
- P3: 10

Average Waiting Time: (6 + 4 + 7)/3= 5.67

Average Completion Time: (30+7+10)/3=15.67

EXAMPLE OF RR WITH TIME QUANTUM = 20

Process	Burst Time
P1	*53*
P2	*8*
P3	*68*
P4	*24*

EXAMPLE OF RR WITH TIME QUANTUM = 20

Process	Burst Time
P1	*~~53~~ ~~33~~ ~~13~~ 0*
P2	*~~8~~ 0*
P3	*~~68~~ ~~48~~ ~~28~~ ~~8~~ 0*
P4	*~~24~~ ~~4~~ 0*

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 28 48 68 88 108 112 125 145 153

EXAMPLE OF RR WITH TIME QUANTUM = 20

Completion Time:

- P1: 125
- P2: 28
- P3: 153
- P4: 112

Process	**Burst Time**
P1	*53*
P2	*8*
P3	*68*
P4	*24*

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 28 48 68 88 108 112 125 145 153

Average Completion Time: (125+28+153+112)/4 = 104.5

EXAMPLE OF RR WITH TIME QUANTUM = 20

Waiting Time:

- For P1:
  - (68-20)+(112-88) = 72

Process	Burst Time
P1	*53*
P2	*8*
P3	*68*
P4	*24*

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 28 48 68 88 108 112 125 145 153

$\underbrace{20 \ldots 68}_{68-20}$ $\underbrace{88 \ldots 112}_{112-88}$

68-20

112-88

EXAMPLE OF RR WITH TIME QUANTUM = 20

Waiting Time:

- For P1:
  - (68-20)+(112-88) = 72
- For P2:
  - 20-0 = 20

Process	**Burst Time**
P1	*53*
P2	*8*
P3	*68*
P4	*24*

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 28 48 68 88 108 112 125 145 153

20 - 0

EXAMPLE OF RR WITH TIME QUANTUM = 20

Waiting Time:

- For P1:
  - (68-20)+(112-88) = 72
- For P2:
  - 20-0 = 20
- For P3
  - (28-0)+(88-48)+(125-108)= 85

Process	**Burst Time**
P1	*53*
P2	*8*
P3	*68*
P4	*24*

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 28 48 68 88 108 112 125 145 153

28 - 0

88 - 48

125 - 108

EXAMPLE OF RR WITH TIME QUANTUM = 20

Waiting Time:

- For P1:
    - (68-20)+(112-88) = 72
- For P2:
    - 20-0 = 20
- For P3
    - (28-0)+(88-48)+(125-108)= 85
- For P4
    - (48-0)+(108-68) = 88

Process	Burst Time
P1	*53*
P2	*8*
P3	*68*
P4	*24*

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 28 48 68 88 108 112 125 145 153

Average Waiting Time: (72+20+85+88)/4 = 66.25

RR SUMMARY

Pros and Cons:

- Better for short jobs (+)
- Fair (+)
- Context-switching time adds up for long jobs (-)

If the chosen quantum is

- too large, response time suffers
- infinite, performance is the same as FCFS
- too small, throughput suffers as percentage of overhead grows

THANK YOU!

QUESTIONS?