Scheduling

CS 241

Motivation

- Desktop Machine: 50-100 processes
  - Each having 5-50 threads
  - **Total**: 250-5000 total threads

- Server Machine: 1000+ processes
  - Each having 5-100 threads
  - Easily over 100,000 total threads!

Meta-Scheduling Strategies

- **Time Slicing**: Give each thread the same time unit, always.
- **Cooperative Multi-tasking**: Ask each thread to yield().
- **Multi-programming**: Evaluate resource usage on each system call, possibly swap out.

Scheduling Strategy

- …but who goes first?
- …and who goes next?

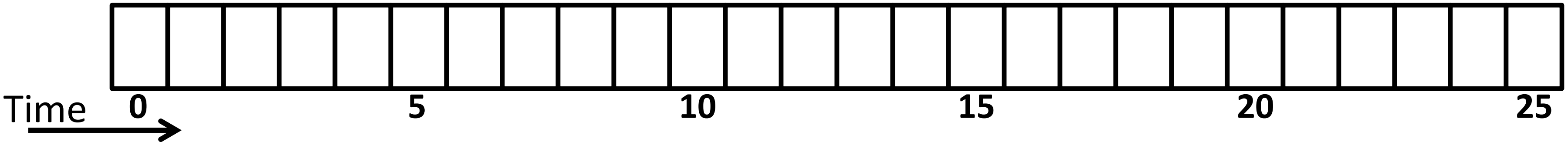
FCFS

- **First Come First Serve** (FCFS): First job to arrive, first to run.

Job ID	Arrival Time	Running Time	Priority
1	0	7	2
2	2	4	3
3	5	11	1 (Important)
4	6	2	4

- **First Come First Serve** (FCFS): First job to arrive, first to run.

Job ID	Arrival Time	Running Time	Priority
1	0	7	2
2	2	4	3
3	5	11	1 (Important)
4	6	2	4



(Non)Preemptive

- Algorithms are broadly classified as preemptive or non-preemptive.
  - **Preemptive**: May swap a job once it has started running.
  - **Non-Preemptive**: Can NOT swap a job once a job has started running.
    - FCFS: Non-Preemptive.

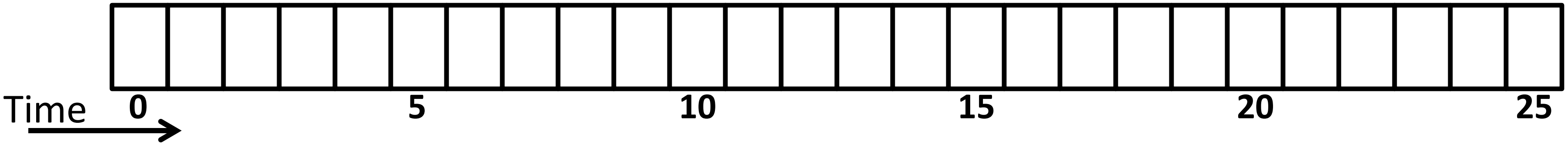
SJF / PSJF

- **Shortest Job First** (SJF): Run the job with the smallest running time, non-preemptively.

- **Preemptive SJF** (PSJF): Always run the job with the *shortest remaining time*, even if this preempts a currently running job.
  - Also known as **Shortest Remaining Time** (SRT)

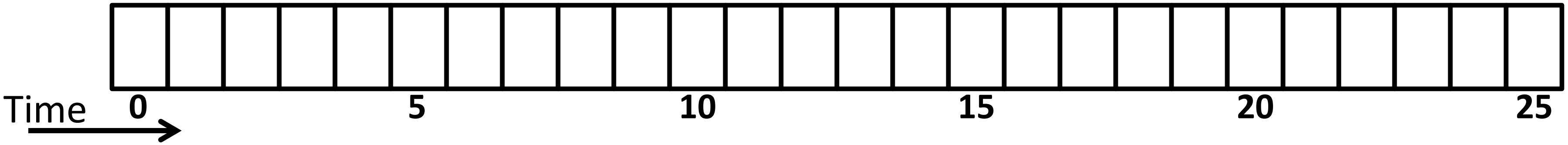
- **Shortest Job First** (SJF)

Job ID	Arrival Time	Running Time	Priority
1	0	7	2
2	2	4	3
3	5	11	1 (Important)
4	6	2	4



- **Preemptive Shortest Job First** (PSJF)

Job ID	Arrival Time	Running Time	Priority
1	0	7	2
2	2	4	3
3	5	11	1 (Important)
4	6	2	4



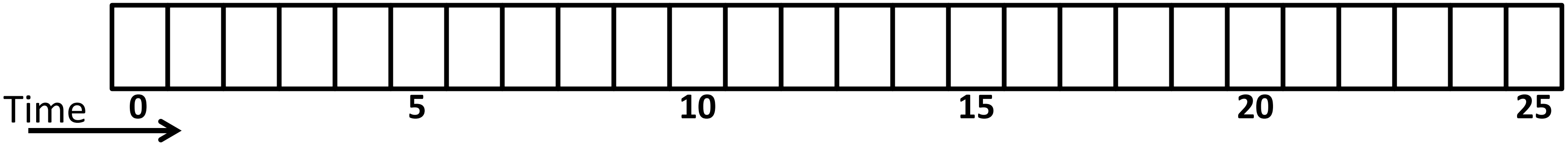
PRI / PPRI

- **Priority** (PRI): Run the most important job first, non-preemptively.

- **Preemptive Priority** (PPRI): Always run the most important job available, even if this preempts a currently running job.

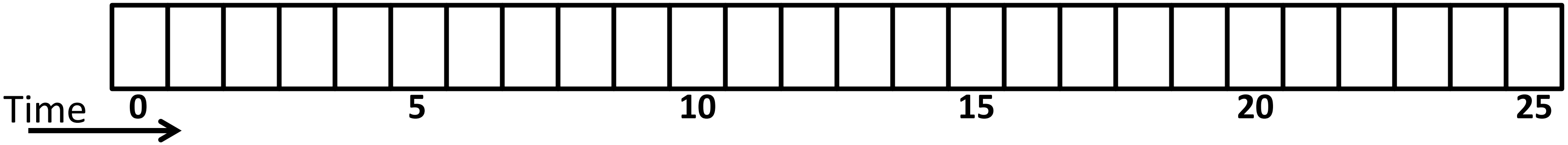
- **Priority** (PRI)

Job ID	Arrival Time	Running Time	Priority
1	0	7	2
2	2	4	3
3	5	11	1 (Important)
4	6	2	4



- **Preemptive Priority** (PPRI)

Job ID	Arrival Time	Running Time	Priority
1	0	7	2
2	2	4	3
3	5	11	1 (Important)
4	6	2	4

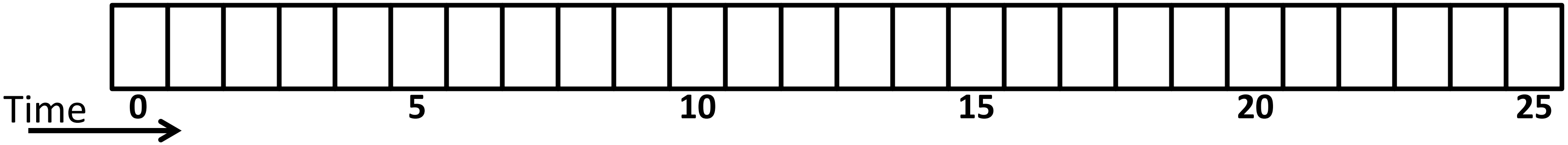


Round Robin (RR)

- **Round Robin** (RR): Run each job for a given time quantum.
  - Use a queue to order jobs.
  - A time quantum must be specified.
    - **RR2** := Time quantum of two (2) time units.

- **Round Robin, q=3** (RR3)

Job ID	Arrival Time	Running Time	Priority
1	0	7	2
2	2	4	3
3	5	11	1 (Important)
4	6	2	4



Round Robin (RR)

- **RR** may act differently depending on the quantum or job size.
  - Very small quantum:
  - Very large quantum:
  - Equal sized jobs:

Algorithm Properties

- **Starvation**: Will every job get scheduled in a fixed amount of time?
  - An algorithm may cause starvation if there exists any scenario where a specific job may never have a chance to run.

FCFS: □	**SJF:** □	**PRI:** □
RR: □	**PSJF:** □	**PPRI:** □

Algorithm Properties

- **Waiting Time**: The amount of time the job spending waiting in the scheduling queue.

- **Response Time**: The amount of time between the arrival of the job and the first time run.

- **Turnaround Time**: The total amount of time the job was in the system (waiting + running).

Algorithm Properties

- Fastest **Average Response Time**?

- Fastest **Average Waiting Time**?

- Fastest **Average Turnaround Time**?

Algorithm Properties

- **Overhead**: How much work does it add to use a given scheduling algorithm?
  - Algorithm Complexity
  - Context Switches

- Algorithm with the highest average overhead?

Scheduling Example

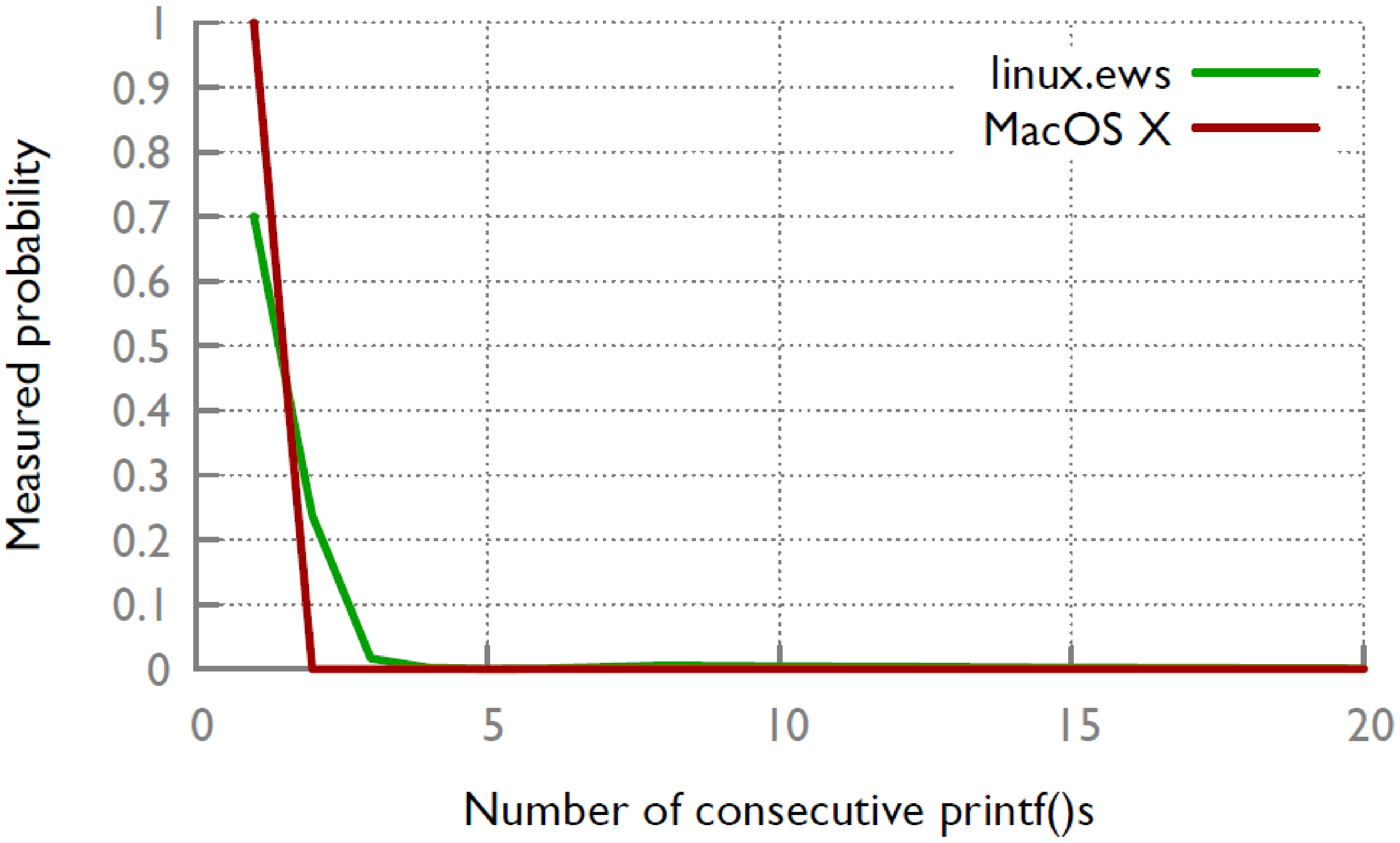
```
/**
 * Print out a identification string forever.
 *
 * @param ptr
 *    Set by main() as a pointer to an (int *) containing
 *    an 0-based ID of the thread.
 */
void * printer_thread( void *ptr )
{
   /* Create the message we will print out */
   int index = *((int *)ptr);
   char *s = asprintf("thread %d\n", index);

   /* Print the string forever. */
   while (1)
       printf("%s", s);

   free(s); return NULL;
}
```

Output

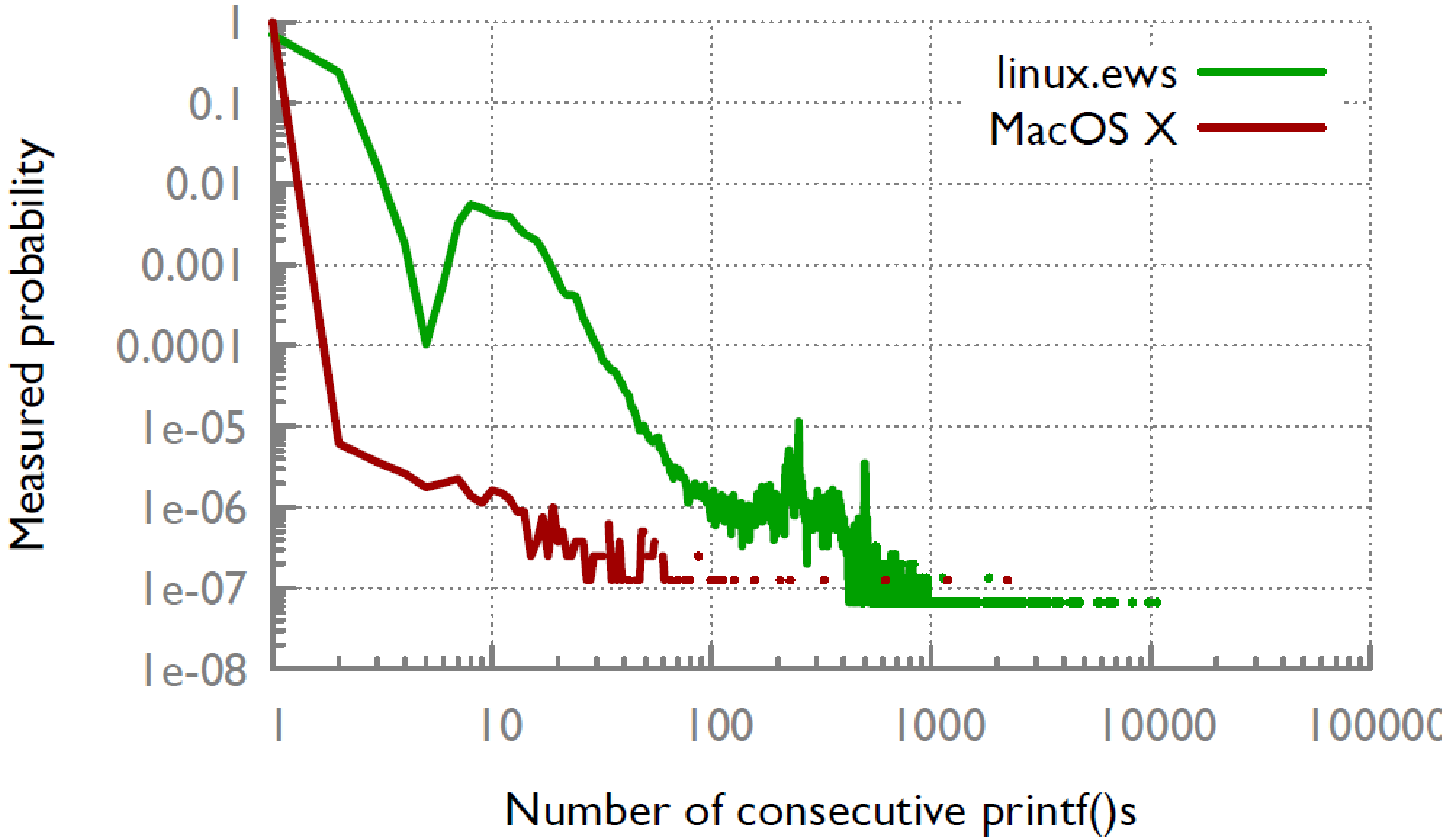
```
thread 1
thread 1
thread 0
thread 0
thread 0
thread 0
thread 0
thread 0
thread 1
thread 1
thread 1
thread 1
thread 1
thread 1
thread 0
thread 0
thread 1
thread 1
…
```

```
thread 1
thread 0
thread 1
thread 0
thread 1
thread 0
thread 1
thread 0
thread 1
thread 0
thread 0
thread 1
thread 0
thread 1
thread 0
thread 1
thread 0
thread 1
…
```

Results


linux.ews
MacOS X
Measured probability
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20
Number of consecutive printf()s

Results



linux.ews
MacOS X
Measured probability
1
0.1
0.01
0.001
0.0001
1e-05
1e-06
1e-07
1e-08
1
10
100
1000
10000
100000
Number of consecutive printf()s

```c
int X = 0; /**< Global variable used for counting. */

/**
 * Increments global variable X by 1 a total of TOTAL times.
 */
void* count_up( void *ptr )
{
       int i = 0;
       for (i=0; i < TOTAL; i++)
             X++;

       return NULL;
}

void main()
{
       pthread_t tid[2]; int i;
       for (i=0; i<2; i++) {
             pthread_create(&tid[i], NULL, count_up, NULL); }
       for (i=0; i<2; i++) {
             pthread_join(tid[i], NULL); }

       printf("%d\n", X);
}
```