Nathan Pelletier
February 19, 2021
AUCSC 380

1.

a. I/O = first half of the program, processor second half

1 job:

Time required for job = N
Average number of jobs per time period = 1/N
Percentage of time processor is active = 50 %

2 jobs:

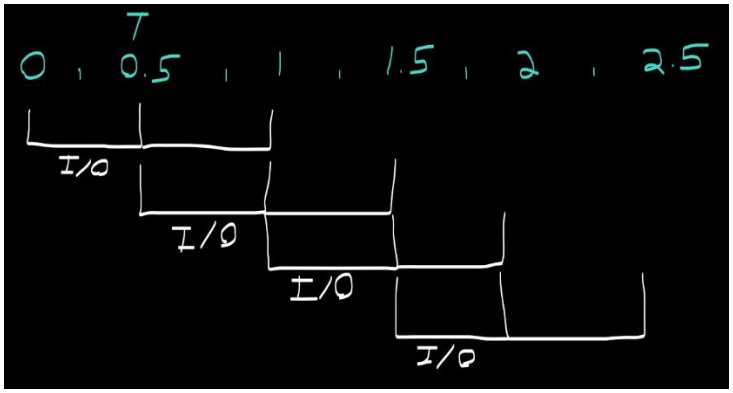
Time required for job = job 1 finishes at 1 N, job 2 finishes at 1.5 N
Average number of jobs per time period = 2/1.5N = 1.33/N
Percentage of time processor is active = 67 %

3 jobs:

Time required for job = job 1 finishes at 1N, 2 at 1.5N, 3 at 2N
Average number of jobs per time period = 3/2N = 1.5/N
Percentage of time processor is active = 75 %

4 jobs:

Time required for job = job 1 finishes at 1N, 2 at 1.5N, 3 at 2N, 4 at 2.5N
Average number of jobs per time period = 4/2.5N = 1.6/N
Percentage of time processor is active = 80 %

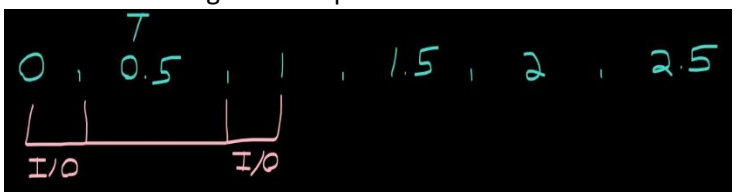


b. I/O first quarter, then half the program goes to the processor, then for the last quarter I/O again

Please note I did not know what to use as time slices, so I used the smallest unit of 1/4 T

1 job:

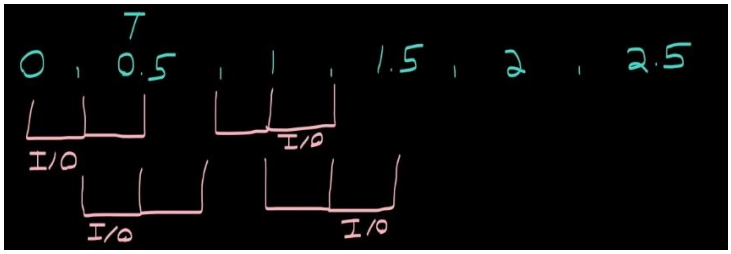
Time required for job = N
Average number of jobs per time period = 1/N
Percentage of time processor is active = 50 %



2 jobs:

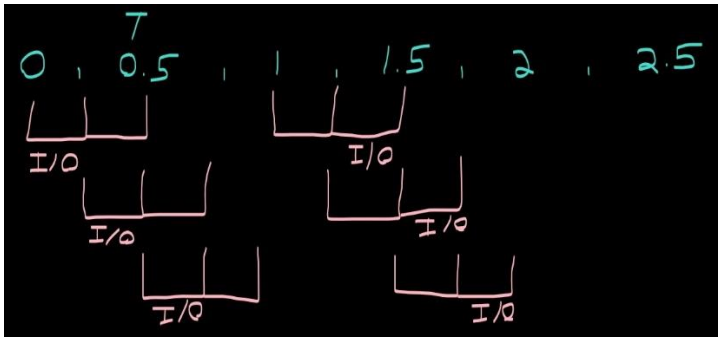
Time required for job = job 1 finished at 1.25 N, job 2 at 1.5 N
Average number of jobs per time period = 2/1.5 N = 1.33/N

Percentage of time processor is active = 67 %



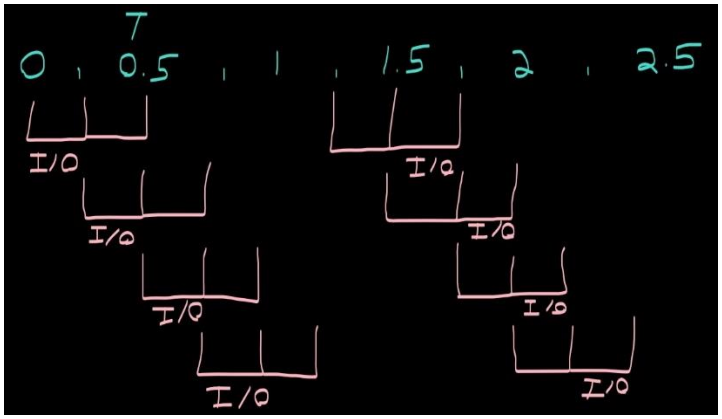
3 jobs:

Time required for job = job 1: 1.5 N, job 2: 1.75 N, job 3: 2N
Average number of jobs per time period = 3/2 N = 1.5/N
Percentage of time processor is active = 75 %



4 jobs:

Time required for job = job 1: 1.75 N, job 2: 2 N, job 3: 2.25 N, job 4: 2.5
Average number of jobs per time period = 4/2.5 N = 1.6/N
Percentage of time processor is active = 80 %



2. System calls are purposeful errors to used halt a program. They can be caused by the users I/O device or by another system calling for resources. System calls allow a computer operating system to maintain responsiveness by switching between critical Kernel tasks and user tasks.
3. OUTPUT:

```
[npelleti@localhost assignment1]$ gcc question3.c
[npelleti@localhost assignment1]$ ./a.out
Child 1 reads this line
Procces 1 ID: 3287
Child 2 reads this line
```

Procces 2 ID: 3288
Child 3 reads this line
[npelleti@localhost assignment1]$ Procces 3 ID: 3289
Child 4 reads this lineProcces 4 ID: 3290

4. exec() : executes a program without further arguments
   execl(): executes a program with a full command line arguments
   execle(): executes a program with a full command line arguments, specify enviroment
   execlp(): executes a program from PATH (searches from current working directory) with arguments
   execlpe(): executes a program from PATH with arguments, specify environment
   execv(): executes a program with array of arguments
   execvp(): executes a program with array of arguments, from PATH
   execvpe(): array of arguments, from PATH, specify environment

   Each function offers a different level of control over the execution of a program. If a programmer was developing a program that leaped into a new program file midway, then maybe they would prefer to customize the environment variables and give a large array of data into the new program. On the other hand, a simplified version that just starts the program may run faster and be easier to program for. In this way we have the options (l: command line arguments, e: environmental variables, p: PATH, v: vector array arguments).

5. Both processes can access the file if reading, however even before executing I expect an error as multiple programs try to write to the same file as the operating system typically prevents this to prevent corruption of data. I've been wrong before though.

   Oddly enough I was wrong. Both processes fight for access to I/O but the queue only gives one access at a time which prevents data corruption. The message comes out clear, even if the lines are mixed up.