Week 2 Homework

Chapter 5:

Question 1

Ran fork.py.

An interesting seed that I found was seed 1, where a forks c and c forks d, but c then exits. I was curious about what happens to d since d did not exit. According to the program tree, d just becomes a child of a.

Text

Description automatically generated with medium confidence

Question 2.

-f, the closer it is to 0, the higher the chance of an exit. The closer “-f” is to one the higher the chance is that there is a fork. I’m predicting that that the closer -f is to one, the tree gets larger, and the closer it is to zero, the more compact it becomes. The more processes that fork, the larger the tree gets and exits decrease the size.

Questions 3.

Text

Description automatically generated

Action 1: a forks b

Action 2: b exits

Action 3: a forks c

Action 4: c forks d

Action 5: a forks e

Question 4:

<https://www.geeksforgeeks.org/zombie-and-orphan-processes-in-c/>

According to this geeks for geeks article, orphaned processes are adopted by the initial process. I initially experience part of this in question 1.

Something interesting that happened when I ran the command was that children that are orphaned get adopted by the main initial process rather than the process that was just the immediate parent of the process. Additionally, when I changed the code so that b was exited rather than C, C,D and E all became children of A.

A picture containing schematic

Description automatically generated

Question 5:

Attempted this several times.

Question 6:

Text

Description automatically generated with medium confidence

Steps:

A + B

A + C

A + D

C + E

B + E

This one was fairly simple to understand. The letters tell you in this program when something was added because they were added sequentially. Additionally, no exits since everything from A to F was in the tree.

Diagram

Description automatically generated with medium confidence

This one is a lot more difficult since B is not there and D is before C when attached to A. I’m guessing that A was attached to B, and C was attached to B. Then d was added to A and e was added to D. Then B was removed and c became a child of A and that’s why it’s slotted after d.

Cases that are fairly easy to tell are where there are no exits and only forks. Exiting removes elements and moves them around making it difficult to tell the sequence of events.

A picture containing graphical user interface

Description automatically generated

This one is almost impossible to tell what happened other than A+D, or child of A + D, or child of a child of A + D.

Chapter 7:

Turnaround = completion – arrival

Response = first run – arrival

Q1

SJF:

All jobs of the same length.

Turnaround time = (200 + 400 + 600) /3 = 400

Response time = (0+200+400) / 3 = 200

FIFO:

Turnaround time = task 1: 200 + task 2: 400 + task 3: 600.

(200 + 400 + 600) / 3 = 400

Response = (0 + 200 + 400) / 3 = 200

Q2:

SJF:

Turnaround time: 100 + 300 + 600 = 1000/ 3

Response time: 0 + 100 + 300 = 400/3

FIFO

Turnaround time: 100 + 300 + 600 = 1000/ 3

Response time: 0 + 100 + 300 = 400/3

Q3:

RR

Estimate Turnaround time: 300 + 500 + 600 / 3 = 1400 /3

Actual turnaround time: 298 + 499 + 600 = 1397 / 3

Response time: 0 + 1 + 2 = 3 / 3 = 1

Q4:

When the first job that arrives is also the shortest job, then FIFO and SJF perform the same.

Q5:

Text

Description automatically generated

When quantum lengths increase to around either the size of the second largest job or larger, the response times of SJF are the same as RR. Type of job does not appear to matter as much as quantum length.

Q6:

Response times increase as the length of jobs increase. Not completely sure what I’m trying to demonstrate. As the length of jobs increase, the shortest one will always run first, but the other jobs must wait longer since the job lengths are increasing.

Text

Description automatically generated

Q7:

As quantum lengths increase, response times also increase. This is because the scheduler is limited in how it can switch processes. It does not go to the next one until a quantum is finished or the process is finished.

Response time equation =( 0 + q1 + q2 + q3 + … + qn ) / n

With n being the number of processes and q being the quantum length.

The program getting the worse response time would be the last program in the sequence:

Worst case for that individual program would be: q(n -1). Each job before it uses a whole quantum up, so that when the final job gets the cpu, it would be quantum \* n-1 for every job before it.