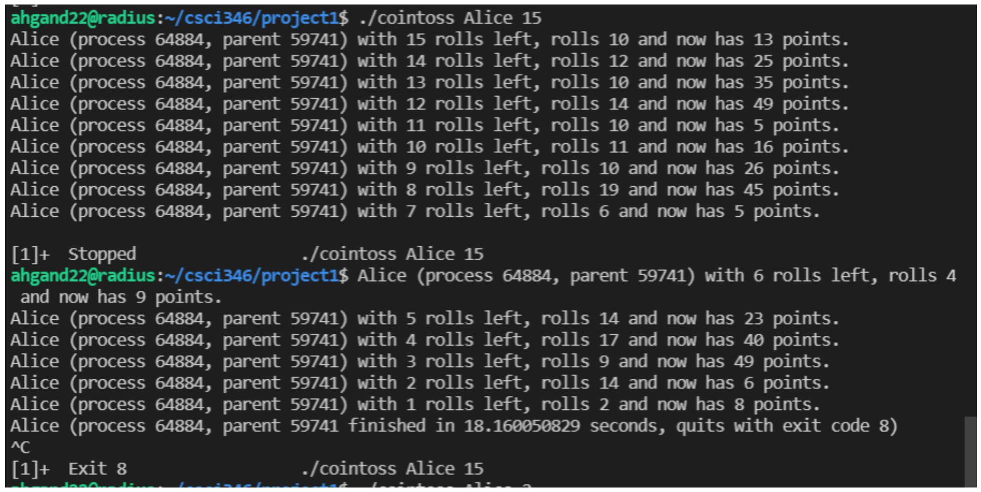
Apurva Gandhi
Professor Walsh
CSCI346: Operating Systems
February 18, 2022

Project 1

Experiments 1

A. From the experiments I performed (data attached as a separate file, “q1_data.pdf”)), it seems to me that there is a pattern in how the operating system is assigning process IDs, and parent IDs. For parent ID, the ID does not seem to change with every call to the cointoss program. Parent ID does changes with the new terminal. Each shell is a program and has its own ID. When a program is executed from the shell, it calls the fork() and execlp() in the background. Therefore, the parent ID is the same, regardless of the number of times the cointoss program is executed. The parent ID will only change when the terminal is changed as it will start a new shell program. – Process IDs seem to have some kind of pattern as well. My observations for the Process IDs imply that there might be some predictability to the pattern for the assigning of the process IDs. They do not seem to be completely random. My data shows that there is some relationship between assigning process IDs and parameters used for the cointoss program. For low value parameter, the difference between two process IDs seems to be much larger compared to the high value parameter to the cointoss programs. During the experiment with two different terimals and different value of parameters, result suggests that with high value parameter, the difference between two process ID is less. Furthermore, running a program into second bash, with same parameters, seem to increase the difference between to process IDs for the second bash compared to first bash. In the data, parameter value 1 and parent ID 41044 has average of 13.6, but parameter value 1 and parent ID 41848 has average of 21.77. One of the reasons for this difference might be that when it sees the input as a larger number, it will know that program will run for a longer time. Therefore, it can assign the number close to the process ID with larger input since the process won’t be rerunning or creating a new process immediately. In contrast, with a smaller input size, it may be that the program will run for a short time and new programs may be created faster and thereby it distributes the process ID more than the larger input.
B. SIGSTOP(19) and SIGCONT(18), SIGXCPU (24), SIGFPE(8)
    a. SIGSTOP(19) and SIGCONT(18): SIGSTOP SIGNAL cannot be caught, blocked, or ignored. Unlike the name suggests, SIGSTOP signal will not terminate a process. Instead, it will pause the process in its current state and keep it in the background. SIGCONT signal will resumes the execution of the process that was stopped. One of the use of this signal is for rsync jobs since it can pause the job, clear up some space on the destination device, and then resume the job. The source rsync process just thinks that the destination rsync process is taking a

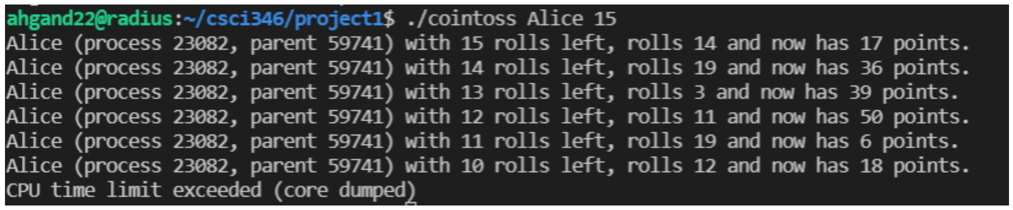
long time to respond.[1] One thing I noticed that I didn’t understand was that after running SIGSTOP and SIGCONT, the process wouldn’t exit on it’s own. I had to exit the process using ctrl+c.



```
ahgand22@radius:~/csci346/project1$ ./cointoss Alice 15
Alice (process 64884, parent 59741) with 15 rolls left, rolls 10 and now has 13 points.
Alice (process 64884, parent 59741) with 14 rolls left, rolls 12 and now has 25 points.
Alice (process 64884, parent 59741) with 13 rolls left, rolls 10 and now has 35 points.
Alice (process 64884, parent 59741) with 12 rolls left, rolls 14 and now has 49 points.
Alice (process 64884, parent 59741) with 11 rolls left, rolls 10 and now has 5 points.
Alice (process 64884, parent 59741) with 10 rolls left, rolls 11 and now has 16 points.
Alice (process 64884, parent 59741) with 9 rolls left, rolls 10 and now has 26 points.
Alice (process 64884, parent 59741) with 8 rolls left, rolls 19 and now has 45 points.
Alice (process 64884, parent 59741) with 7 rolls left, rolls 6 and now has 5 points.

[1]+  Stopped                 ./cointoss Alice 15
ahgand22@radius:~/csci346/project1$ Alice (process 64884, parent 59741) with 6 rolls left, rolls 4
 and now has 9 points.
Alice (process 64884, parent 59741) with 5 rolls left, rolls 14 and now has 23 points.
Alice (process 64884, parent 59741) with 4 rolls left, rolls 17 and now has 40 points.
Alice (process 64884, parent 59741) with 3 rolls left, rolls 9 and now has 49 points.
Alice (process 64884, parent 59741) with 2 rolls left, rolls 14 and now has 6 points.
Alice (process 64884, parent 59741) with 1 rolls left, rolls 2 and now has 8 points.
Alice (process 64884, parent 59741 finished in 18.160050829 seconds, quits with exit code 8)
^C
[1]+  Exit 8                  ./cointoss Alice 15
```

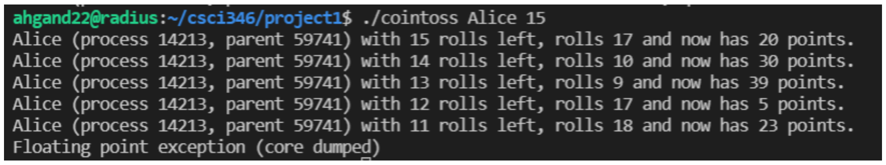
b. SIGXCPU (24): The SIGXCPU a signal is sent to a process when it has used up the CPU for a duration that exceeds a certain predetermined user-settable value. The arrival of a SIGXCPU signal provides the receiving process a chance to quickly save any intermediate results and to exit gracefully, before it is terminated by the operating system using the SIGKILL signal.[2]



```
ahgand22@radius:~/csci346/project1$ ./cointoss Alice 15
Alice (process 23082, parent 59741) with 15 rolls left, rolls 14 and now has 17 points.
Alice (process 23082, parent 59741) with 14 rolls left, rolls 19 and now has 36 points.
Alice (process 23082, parent 59741) with 13 rolls left, rolls 3 and now has 39 points.
Alice (process 23082, parent 59741) with 12 rolls left, rolls 11 and now has 50 points.
Alice (process 23082, parent 59741) with 11 rolls left, rolls 19 and now has 6 points.
Alice (process 23082, parent 59741) with 10 rolls left, rolls 12 and now has 18 points.
CPU time limit exceeded (core dumped)
```

c. SIGFPE: The SIGFPE signal is sent to a process when it executes an erroneous arithmetic operation, such as division by zero.[3] There can be many different arithmetic errors and so this signal will help with that. For example, If a program stores integer data in a location which is then used in a floating-point operation, this often causes an “invalid operation” exception, because the processor cannot recognize the data as a floating-point number.[4]

[1]https://major.io/2009/06/15/two-great-signals-sigstop-and-sigcont/#:~:text=the%20SIGCONT%20signal.-,SIGSTOP%20and%20SIGCONT%20are%20used%20for%20job%20control%20in%20the,up%20where%20you%20left%20off%E2%80%9D.
[2]https://dsa.cs.tsinghua.edu.cn/oj/static/unix_signal.html#:~:text=The%20SIGALRM%2C%20SIGVTALRM%20and%20SIGPROF,used%20by%20the%20process%20elapses.
[3]https://dsa.cs.tsinghua.edu.cn/oj/static/unix_signal.html#:~:text=The%20SIGALRM%2C%20SIGVTALRM%20and%20SIGPROF,used%20by%20the%20process%20elapses.
[4] https://www.gnu.org/software/libc/manual/html_node/Program-Error-Signals.html



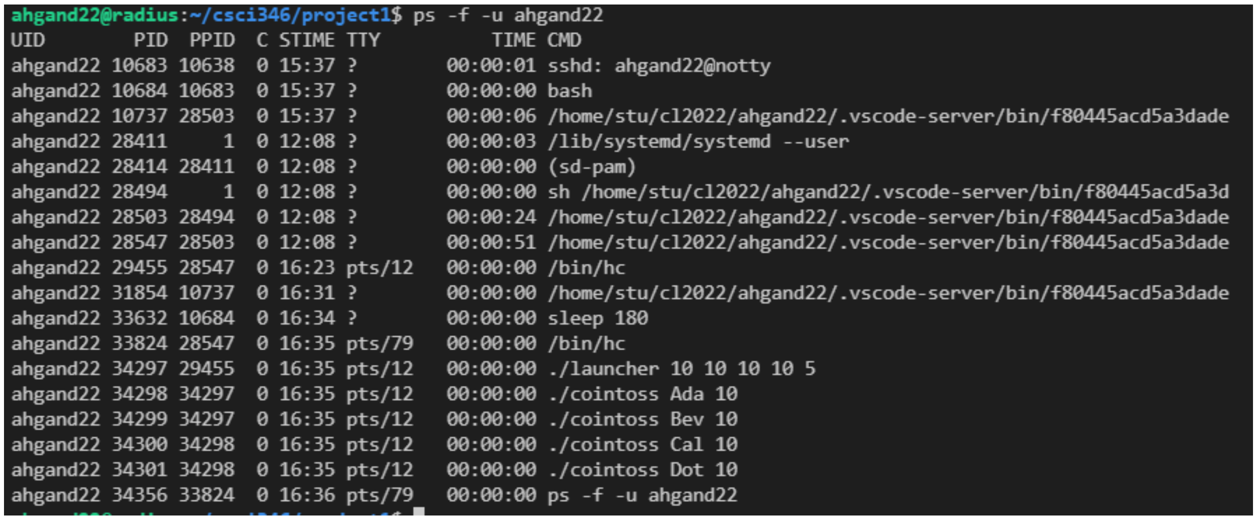
```
ahgand22@radius:~/csci346/project1$ ./cointoss Alice 15
Alice (process 14213, parent 59741) with 15 rolls left, rolls 17 and now has 20 points.
Alice (process 14213, parent 59741) with 14 rolls left, rolls 10 and now has 30 points.
Alice (process 14213, parent 59741) with 13 rolls left, rolls 9 and now has 39 points.
Alice (process 14213, parent 59741) with 12 rolls left, rolls 17 and now has 5 points.
Alice (process 14213, parent 59741) with 11 rolls left, rolls 18 and now has 23 points.
Floating point exception (core dumped)
```

C. While the bash on VS Code limits the output, the lowest ID number I can see is 153. Root started the process. When I tried to kill that process, it printed that hc: kill: (153) - Operation not permitted.

D. When trying to kill the process of other users, it does not allow the process to be killed. It provides a similar error of operation not permitted. hc: kill: (57980) - Operation not permitted. Something I found interesting was that process does kill itself. For example, when I tried killing the bash process within bash, it killed itself successfully.

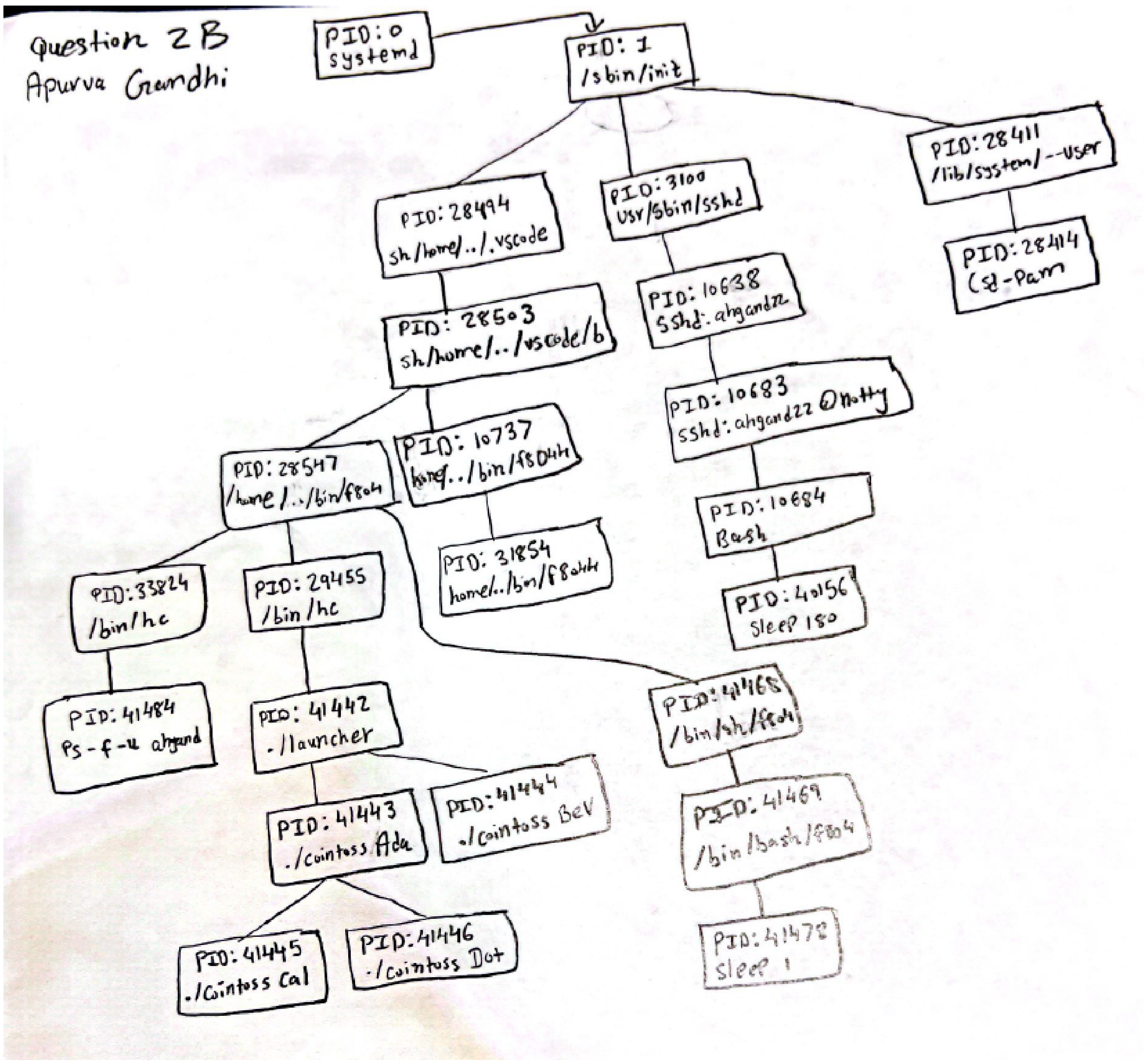
Experiments 2

A.



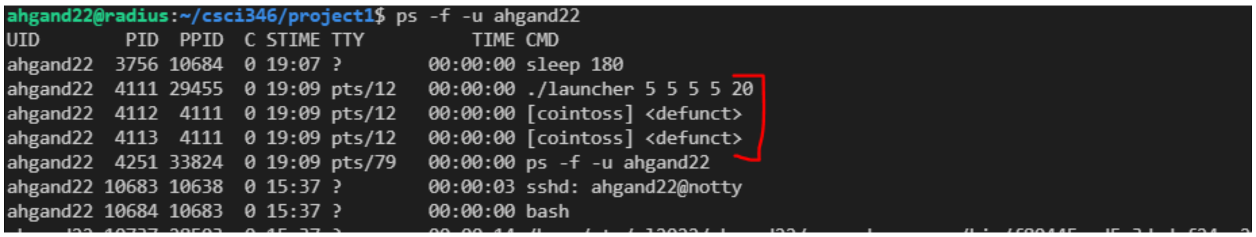
```
ahgand22@radius:~/csci346/project1$ ps -f -u ahgand22
UID        PID  PPID  C STIME TTY          TIME CMD
ahgand22 10683 10638  0 15:37 ?        00:00:01 sshd: ahgand22@notty
ahgand22 10684 10683  0 15:37 ?        00:00:00 bash
ahgand22 10737 28503  0 15:37 ?        00:00:06 /home/stu/cl2022/ahgand22/.vscode-server/bin/f80445acd5a3dade
ahgand22 28411     1  0 12:08 ?        00:00:03 /lib/systemd/systemd --user
ahgand22 28414 28411  0 12:08 ?        00:00:00 (sd-pam)
ahgand22 28494     1  0 12:08 ?        00:00:00 sh /home/stu/cl2022/ahgand22/.vscode-server/bin/f80445acd5a3d
ahgand22 28503 28494  0 12:08 ?        00:00:24 /home/stu/cl2022/ahgand22/.vscode-server/bin/f80445acd5a3dade
ahgand22 28547 28503  0 12:08 ?        00:00:51 /home/stu/cl2022/ahgand22/.vscode-server/bin/f80445acd5a3dade
ahgand22 29455 28547  0 16:23 pts/12   00:00:00 /bin/hc
ahgand22 31854 10737  0 16:31 ?        00:00:00 /home/stu/cl2022/ahgand22/.vscode-server/bin/f80445acd5a3dade
ahgand22 33632 10684  0 16:34 ?        00:00:00 sleep 180
ahgand22 33824 28547  0 16:35 pts/79   00:00:00 /bin/hc
ahgand22 34297 29455  0 16:35 pts/12   00:00:00 ./launcher 10 10 10 10 5
ahgand22 34298 34297  0 16:35 pts/12   00:00:00 ./cointoss Ada 10
ahgand22 34299 34297  0 16:35 pts/12   00:00:00 ./cointoss Bev 10
ahgand22 34300 34298  0 16:35 pts/12   00:00:00 ./cointoss Cal 10
ahgand22 34301 34298  0 16:35 pts/12   00:00:00 ./cointoss Dot 10
ahgand22 34356 33824  0 16:36 pts/79   00:00:00 ps -f -u ahgand22
```

B.


Question 2B
Apurva Gandhi
PID: 0 systemd
PID: 1 /sbin/init
PID: 28494 sh /home/../.vscode
PID: 3100 usr/sbin/sshd
PID: 28411 /lib/systen/--user
PID: 28414 (sd-pam
PID: 28503 sh/home/../vscode/b
PID: 10638 sshd: ahgandzz
PID: 10683 sshd: ahgand22 @notty
PID: 28547 /home/../bin/f804
PID: 10737 home/../bin/f804h
PID: 10684 Bash
PID: 31854 home/../bin/f804h
PID: 40156 Sleep 180
PID: 33824 /bin/hc
PID: 29455 /bin/hc
PID: 41468 /bin/sh/f804
PID: 41484 ps -f -u ahgand
PID: 41442 ./launcher
PID: 41443 ./cointoss Ada
PID: 41444 ./cointoss Bev
PID: 41469 /bin/bash/f804
PID: 41445 ./cointoss Cal
PID: 41446 ./cointoss Dot
PID: 41478 sleep 1

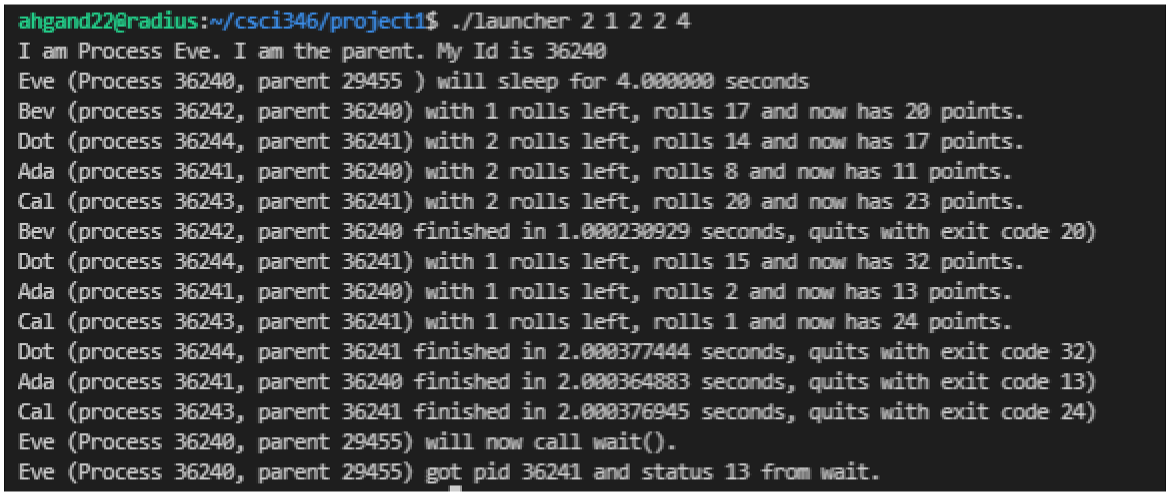
C. If the child is terminated before the parent performs the wait() call, “performing a wait allows the system to release the resources associated with the child.” If a wait is not performed, then the terminated child remains in a "zombie" state.” In the zombie state, the kernel maintains a minimal set of information such as PID, termination status, and resource usage information. This information will allow the parent to perform a wait to obtain information about the child. Removing zombies is important because it will consume a slot in the kernel process table and may cause the table to get filled up, resulting in an inability to create further processes. In this case, if a parent is terminated, then its zombie children are adopted by init, which performs a wait to remove the zombie children.[5]

I am Process Eve. I am the parent. My Id is 4111
I am Process Ada. My Id is 4112
I am Process Bev. My Id is 4113
I am Process Cal. My Id is 4114.
I am Process Dot. My Id is 4115
If all of the children are terminated before the parent calls wait(), the direct children will be in zombie (defunct) state. The grandchildren are considered orphaned since their parents are dead. Therefore, they will be taken care of by init process. In the image below, Ada and Bev are in zombie (defunct) state and Cal and Dot are takencare of by init.



```
ahgand22@radius:~/csci346/project1$ ps -f -u ahgand22
UID        PID  PPID  C STIME TTY          TIME CMD
ahgand22  3756 10684  0 19:07 ?        00:00:00 sleep 180
ahgand22  4111 29455  0 19:09 pts/12   00:00:00 ./launcher 5 5 5 5 20
ahgand22  4112  4111  0 19:09 pts/12   00:00:00 [cointoss] <defunct>
ahgand22  4113  4111  0 19:09 pts/12   00:00:00 [cointoss] <defunct>
ahgand22  4251 33824  0 19:09 pts/79   00:00:00 ps -f -u ahgand22
ahgand22 10683 10638  0 15:37 ?        00:00:03 sshd: ahgand22@notty
ahgand22 10684 10683  0 15:37 ?        00:00:00 bash
```

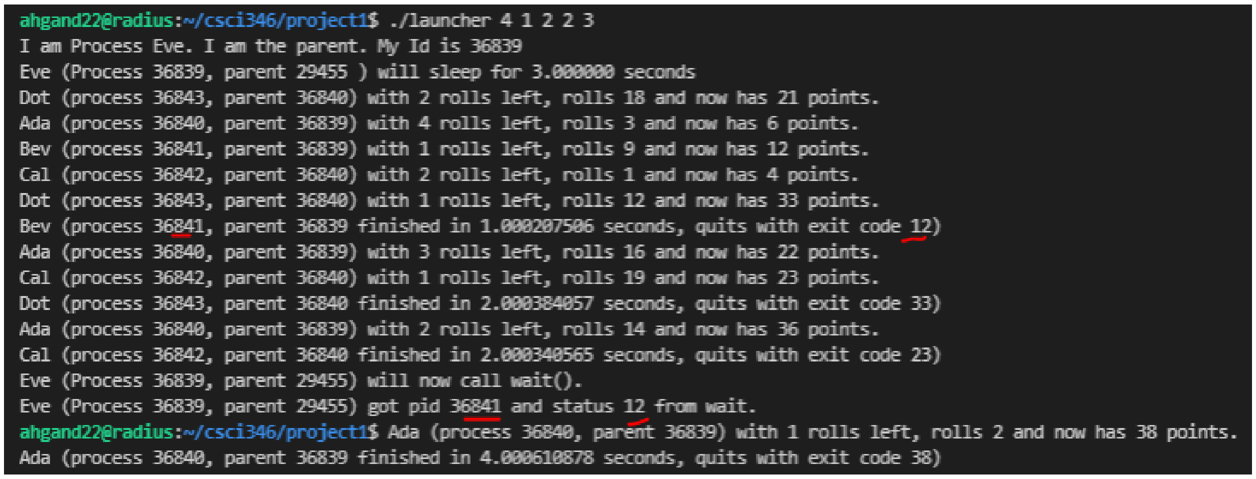
If all of the children are terminated, the wait call returns immediately. Furthermore, first forked process is returned as the result of wait().



```
ahgand22@radius:~/csci346/project1$ ./launcher 2 1 2 2 4
I am Process Eve. I am the parent. My Id is 36240
Eve (Process 36240, parent 29455 ) will sleep for 4.000000 seconds
Bev (process 36242, parent 36240) with 1 rolls left, rolls 17 and now has 20 points.
Dot (process 36244, parent 36241) with 2 rolls left, rolls 14 and now has 17 points.
Ada (process 36241, parent 36240) with 2 rolls left, rolls 8 and now has 11 points.
Cal (process 36243, parent 36241) with 2 rolls left, rolls 20 and now has 23 points.
Bev (process 36242, parent 36240 finished in 1.000230929 seconds, quits with exit code 20)
Dot (process 36244, parent 36241) with 1 rolls left, rolls 15 and now has 32 points.
Ada (process 36241, parent 36240) with 1 rolls left, rolls 2 and now has 13 points.
Cal (process 36243, parent 36241) with 1 rolls left, rolls 1 and now has 24 points.
Dot (process 36244, parent 36241 finished in 2.000377444 seconds, quits with exit code 32)
Ada (process 36241, parent 36240 finished in 2.000364883 seconds, quits with exit code 13)
Cal (process 36243, parent 36241 finished in 2.000376945 seconds, quits with exit code 24)
Eve (Process 36240, parent 29455) will now call wait().
Eve (Process 36240, parent 29455) got pid 36241 and status 13 from wait.
```

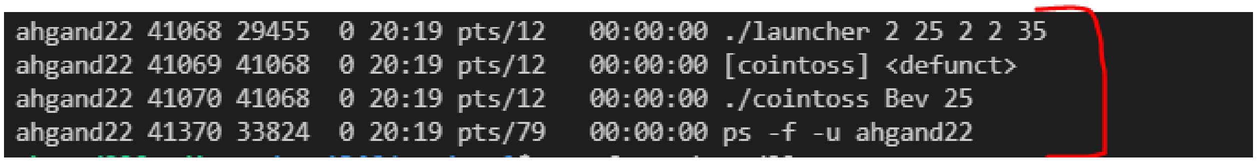
D. If only some of the direct children have already terminated before the parent process calls wait(), then wait will get value from the child who is terminated. Children who is not terminated will continue to execute. In the example below, Bev is terminated when wait() is called and therefore the value of Bev is printed after wait().

[5] https://linux.die.net/man/2/wait



```
ahgand22@radius:~/csci346/project1$ ./launcher 4 1 2 2 3
I am Process Eve. I am the parent. My Id is 36839
Eve (Process 36839, parent 29455 ) will sleep for 3.000000 seconds
Dot (process 36843, parent 36840) with 2 rolls left, rolls 18 and now has 21 points.
Ada (process 36840, parent 36839) with 4 rolls left, rolls 3 and now has 6 points.
Bev (process 36841, parent 36839) with 1 rolls left, rolls 9 and now has 12 points.
Cal (process 36842, parent 36840) with 2 rolls left, rolls 1 and now has 4 points.
Dot (process 36843, parent 36840) with 1 rolls left, rolls 12 and now has 33 points.
Bev (process 36841, parent 36839 finished in 1.000207506 seconds, quits with exit code 12)
Ada (process 36840, parent 36839) with 3 rolls left, rolls 16 and now has 22 points.
Cal (process 36842, parent 36840) with 1 rolls left, rolls 19 and now has 23 points.
Dot (process 36843, parent 36840 finished in 2.000384057 seconds, quits with exit code 33)
Ada (process 36840, parent 36839) with 2 rolls left, rolls 14 and now has 36 points.
Cal (process 36842, parent 36840 finished in 2.000340565 seconds, quits with exit code 23)
Eve (Process 36839, parent 29455) will now call wait().
Eve (Process 36839, parent 29455) got pid 36841 and status 12 from wait.
ahgand22@radius:~/csci346/project1$ Ada (process 36840, parent 36839) with 1 rolls left, rolls 2 and now has 38 points.
Ada (process 36840, parent 36839 finished in 4.000610878 seconds, quits with exit code 38)
```

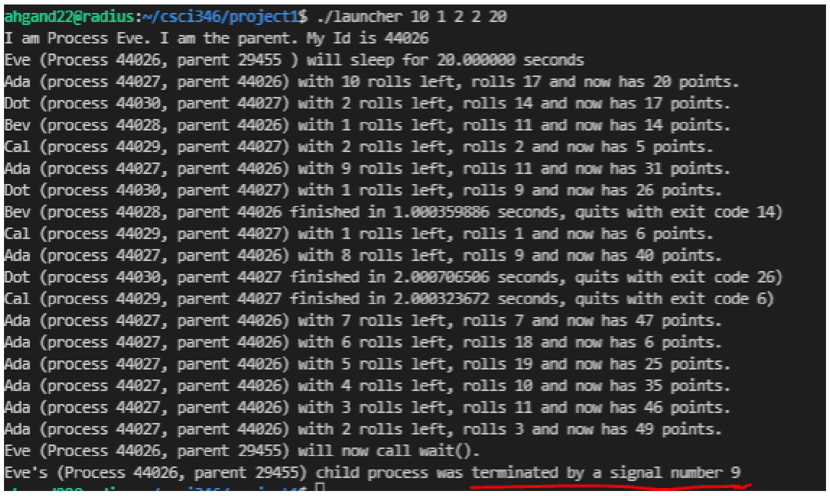
Furthermore, if only some of the direct children have already terminated before the parent process calls, then only that children will be in the zombie state. For example, only Ada is in the zombie state. Again, Cal and Dot are adopted by init. (Both images were not executed at the same time)



```
ahgand22 41068 29455  0 20:19 pts/12   00:00:00 ./launcher 2 25 2 2 35
ahgand22 41069 41068  0 20:19 pts/12   00:00:00 [cointoss] <defunct>
ahgand22 41070 41068  0 20:19 pts/12   00:00:00 ./cointoss Bev 25
ahgand22 41370 33824  0 20:19 pts/79   00:00:00 ps -f -u ahgand22
```

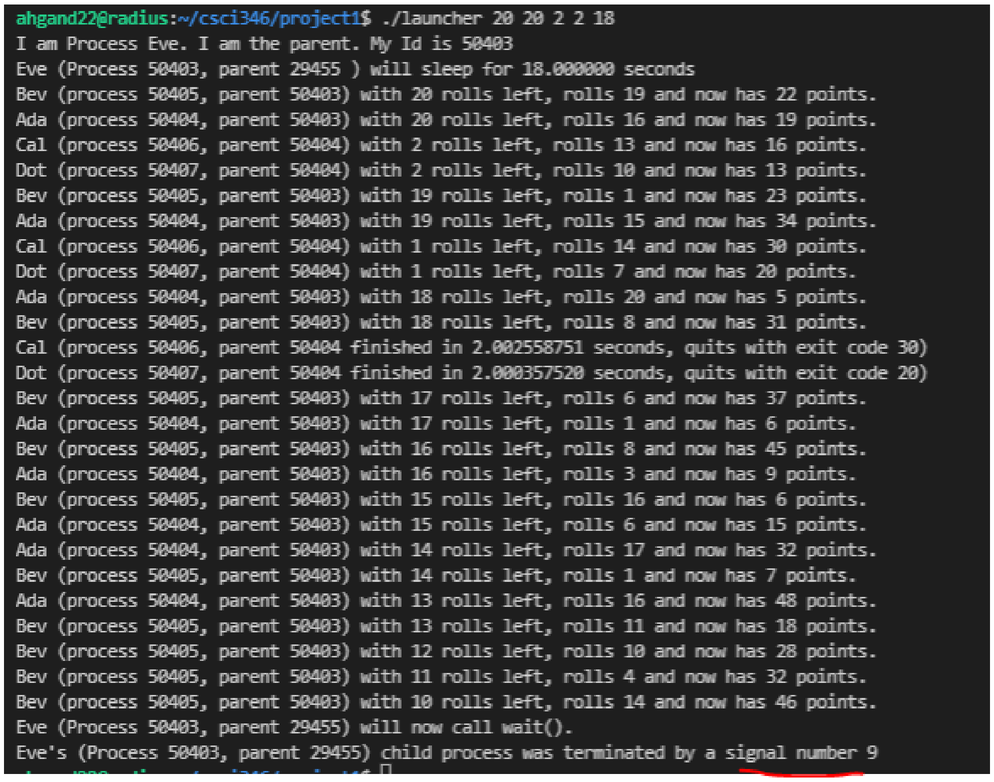
E. No, the parent would not notice in any way that child has been killed before the parent process calls the wait(). The parent does notice that the child has been killed after calling wait(). It reliably distinguishes between a child that exits normally and one that was killed through the signal. WIFEXITED returns true if the child terminated normally and so if it returns false, then we can check for WIFSIGNALED which returns true if the child process was terminated by a signal. Finally, WTERMSIG returns the number of the signal that caused the child process to terminate.

Case 1: All children are terminated except 1. The alive child is killed by signal. Ada is killed by signal here and all other child are terminated. This can be checked with wait().



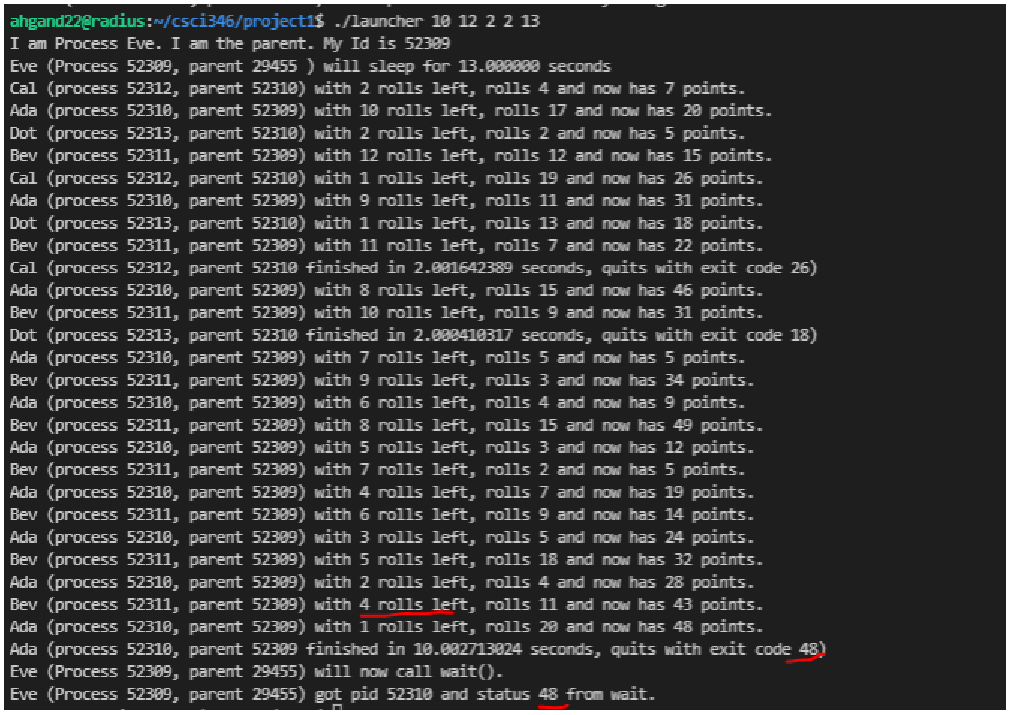
```
ahgand22@radius:~/csci346/project1$ ./launcher 10 1 2 2 20
I am Process Eve. I am the parent. My Id is 44026
Eve (Process 44026, parent 29455 ) will sleep for 20.000000 seconds
Ada (process 44027, parent 44026) with 10 rolls left, rolls 17 and now has 20 points.
Dot (process 44030, parent 44027) with 2 rolls left, rolls 14 and now has 17 points.
Bev (process 44028, parent 44026) with 1 rolls left, rolls 11 and now has 14 points.
Cal (process 44029, parent 44027) with 2 rolls left, rolls 2 and now has 5 points.
Ada (process 44027, parent 44026) with 9 rolls left, rolls 11 and now has 31 points.
Dot (process 44030, parent 44027) with 1 rolls left, rolls 9 and now has 26 points.
Bev (process 44028, parent 44026 finished in 1.000359886 seconds, quits with exit code 14)
Cal (process 44029, parent 44027) with 1 rolls left, rolls 1 and now has 6 points.
Ada (process 44027, parent 44026) with 8 rolls left, rolls 9 and now has 40 points.
Dot (process 44030, parent 44027 finished in 2.000706506 seconds, quits with exit code 26)
Cal (process 44029, parent 44027 finished in 2.000323672 seconds, quits with exit code 6)
Ada (process 44027, parent 44026) with 7 rolls left, rolls 7 and now has 47 points.
Ada (process 44027, parent 44026) with 6 rolls left, rolls 18 and now has 6 points.
Ada (process 44027, parent 44026) with 5 rolls left, rolls 19 and now has 25 points.
Ada (process 44027, parent 44026) with 4 rolls left, rolls 10 and now has 35 points.
Ada (process 44027, parent 44026) with 3 rolls left, rolls 11 and now has 46 points.
Ada (process 44027, parent 44026) with 2 rolls left, rolls 3 and now has 49 points.
Eve (Process 44026, parent 29455) will now call wait().
Eve's (Process 44026, parent 29455) child process was terminated by a signal number 9
```

Case 2:If both children are killed by a signal.



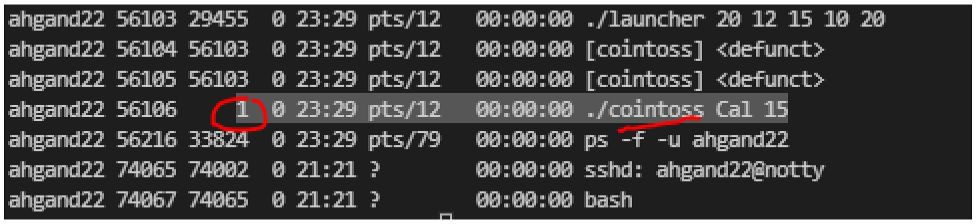
```
ahgand22@radius:~/csci346/project1$ ./launcher 20 20 2 2 18
I am Process Eve. I am the parent. My Id is 50403
Eve (Process 50403, parent 29455 ) will sleep for 18.000000 seconds
Bev (process 50405, parent 50403) with 20 rolls left, rolls 19 and now has 22 points.
Ada (process 50404, parent 50403) with 20 rolls left, rolls 16 and now has 19 points.
Cal (process 50406, parent 50404) with 2 rolls left, rolls 13 and now has 16 points.
Dot (process 50407, parent 50404) with 2 rolls left, rolls 10 and now has 13 points.
Bev (process 50405, parent 50403) with 19 rolls left, rolls 1 and now has 23 points.
Ada (process 50404, parent 50403) with 19 rolls left, rolls 15 and now has 34 points.
Cal (process 50406, parent 50404) with 1 rolls left, rolls 14 and now has 30 points.
Dot (process 50407, parent 50404) with 1 rolls left, rolls 7 and now has 20 points.
Ada (process 50404, parent 50403) with 18 rolls left, rolls 20 and now has 5 points.
Bev (process 50405, parent 50403) with 18 rolls left, rolls 8 and now has 31 points.
Cal (process 50406, parent 50404 finished in 2.002558751 seconds, quits with exit code 30)
Dot (process 50407, parent 50404 finished in 2.000357520 seconds, quits with exit code 20)
Bev (process 50405, parent 50403) with 17 rolls left, rolls 6 and now has 37 points.
Ada (process 50404, parent 50403) with 17 rolls left, rolls 1 and now has 6 points.
Bev (process 50405, parent 50403) with 16 rolls left, rolls 8 and now has 45 points.
Ada (process 50404, parent 50403) with 16 rolls left, rolls 3 and now has 9 points.
Bev (process 50405, parent 50403) with 15 rolls left, rolls 16 and now has 6 points.
Ada (process 50404, parent 50403) with 15 rolls left, rolls 6 and now has 15 points.
Ada (process 50404, parent 50403) with 14 rolls left, rolls 17 and now has 32 points.
Bev (process 50405, parent 50403) with 14 rolls left, rolls 1 and now has 7 points.
Ada (process 50404, parent 50403) with 13 rolls left, rolls 16 and now has 48 points.
Bev (process 50405, parent 50403) with 13 rolls left, rolls 11 and now has 18 points.
Bev (process 50405, parent 50403) with 12 rolls left, rolls 10 and now has 28 points.
Bev (process 50405, parent 50403) with 11 rolls left, rolls 4 and now has 32 points.
Bev (process 50405, parent 50403) with 10 rolls left, rolls 14 and now has 46 points.
Eve (Process 50403, parent 29455) will now call wait().
Eve's (Process 50403, parent 29455) child process was terminated by a signal number 9
```

Case 3: If a first forked child is terminated with exit() call and second forked() child is killed by a signal, wait to get the result of the first fork() child and does not detect if the child was killed by a signal.



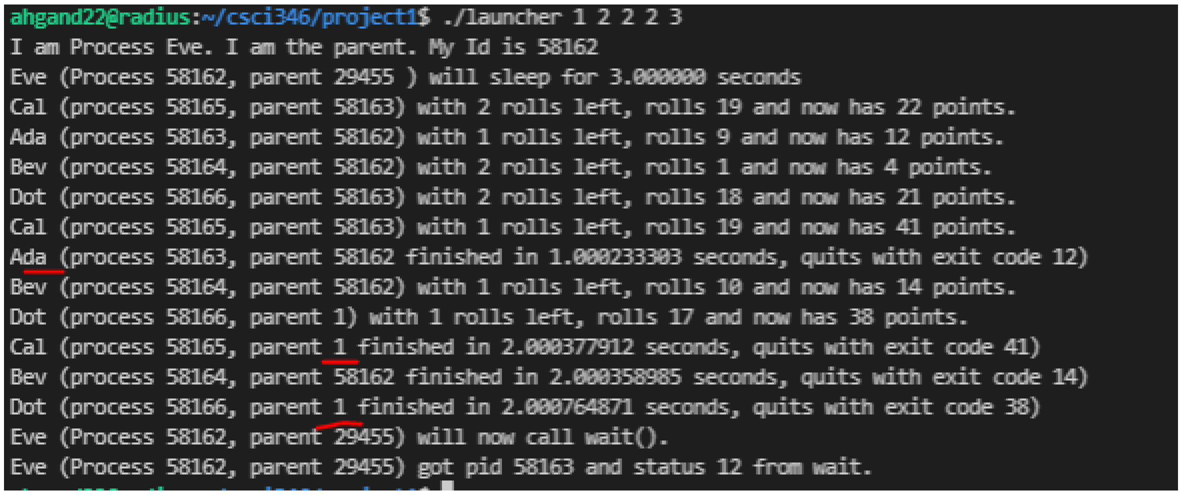
```
ahgand22@radius:~/csci346/project1$ ./launcher 10 12 2 2 13
I am Process Eve. I am the parent. My Id is 52309
Eve (Process 52309, parent 29455 ) will sleep for 13.000000 seconds
Cal (process 52312, parent 52310) with 2 rolls left, rolls 4 and now has 7 points.
Ada (process 52310, parent 52309) with 10 rolls left, rolls 17 and now has 20 points.
Dot (process 52313, parent 52310) with 2 rolls left, rolls 2 and now has 5 points.
Bev (process 52311, parent 52309) with 12 rolls left, rolls 12 and now has 15 points.
Cal (process 52312, parent 52310) with 1 rolls left, rolls 19 and now has 26 points.
Ada (process 52310, parent 52309) with 9 rolls left, rolls 11 and now has 31 points.
Dot (process 52313, parent 52310) with 1 rolls left, rolls 13 and now has 18 points.
Bev (process 52311, parent 52309) with 11 rolls left, rolls 7 and now has 22 points.
Cal (process 52312, parent 52310 finished in 2.001642389 seconds, quits with exit code 26)
Ada (process 52310, parent 52309) with 8 rolls left, rolls 15 and now has 46 points.
Bev (process 52311, parent 52309) with 10 rolls left, rolls 9 and now has 31 points.
Dot (process 52313, parent 52310 finished in 2.000410317 seconds, quits with exit code 18)
Ada (process 52310, parent 52309) with 7 rolls left, rolls 5 and now has 5 points.
Bev (process 52311, parent 52309) with 9 rolls left, rolls 3 and now has 34 points.
Ada (process 52310, parent 52309) with 6 rolls left, rolls 4 and now has 9 points.
Bev (process 52311, parent 52309) with 8 rolls left, rolls 15 and now has 49 points.
Ada (process 52310, parent 52309) with 5 rolls left, rolls 3 and now has 12 points.
Bev (process 52311, parent 52309) with 7 rolls left, rolls 2 and now has 5 points.
Ada (process 52310, parent 52309) with 4 rolls left, rolls 7 and now has 19 points.
Bev (process 52311, parent 52309) with 6 rolls left, rolls 9 and now has 14 points.
Ada (process 52310, parent 52309) with 3 rolls left, rolls 5 and now has 24 points.
Bev (process 52311, parent 52309) with 5 rolls left, rolls 18 and now has 32 points.
Ada (process 52310, parent 52309) with 2 rolls left, rolls 4 and now has 28 points.
Bev (process 52311, parent 52309) with 4 rolls left, rolls 11 and now has 43 points.
Ada (process 52310, parent 52309) with 1 rolls left, rolls 20 and now has 48 points.
Ada (process 52310, parent 52309 finished in 10.002713024 seconds, quits with exit code 48)
Eve (Process 52309, parent 29455) will now call wait().
Eve (Process 52309, parent 29455) got pid 52310 and status 48 from wait.
```

F. The exit values of the descendants of a killed process are lost. Descendants continue running but their parent is changed to the init (parent Id = 1). Their values are just lost once they finish their execution. In the example below, Ada is killed but Cal continues doing its execution. Ada and Bev are in a zombie state and Dot is taken care of by init because it became orphaned when Ada was killed. The values are probably accessed through the init process but not access to the original parent.



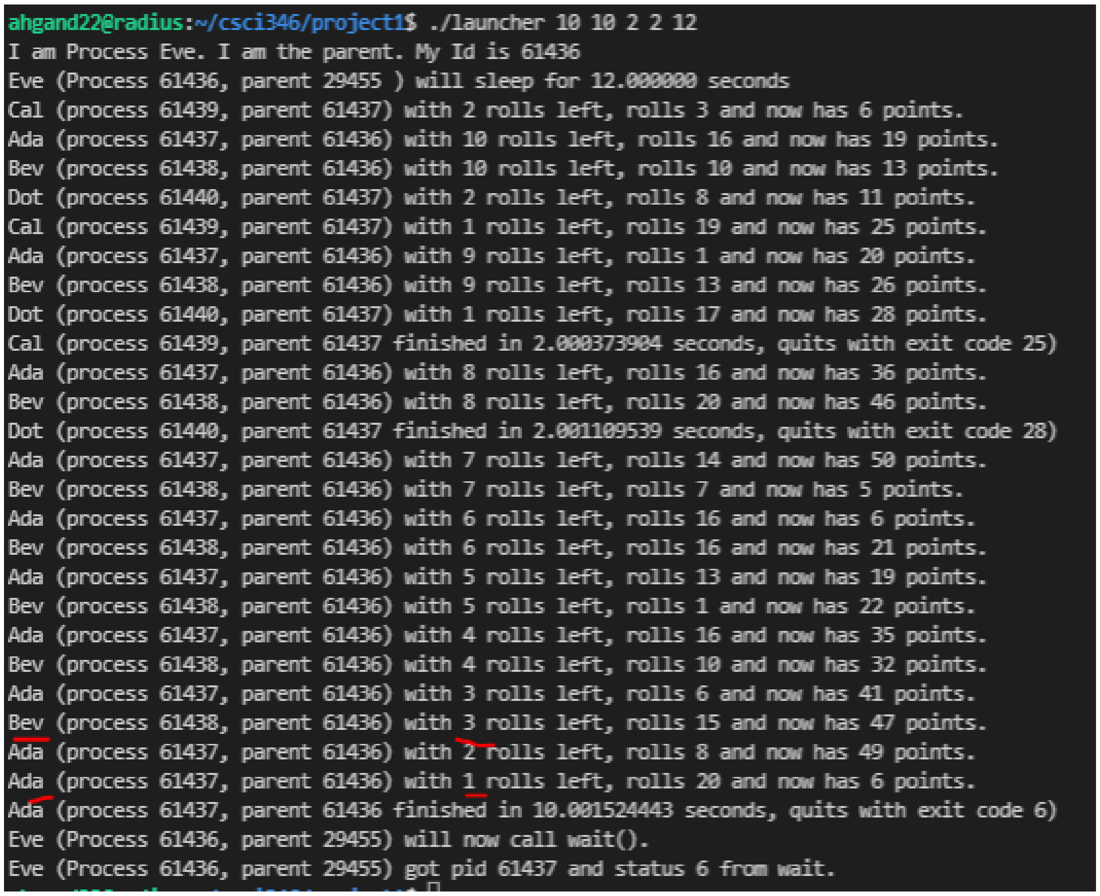
```
ahgand22 56103 29455  0 23:29 pts/12   00:00:00 ./launcher 20 12 15 10 20
ahgand22 56104 56103  0 23:29 pts/12   00:00:00 [cointoss] <defunct>
ahgand22 56105 56103  0 23:29 pts/12   00:00:00 [cointoss] <defunct>
ahgand22 56106     1  0 23:29 pts/12   00:00:00 ./cointoss Cal 15
ahgand22 56216 33824  0 23:29 pts/79   00:00:00 ps -f -u ahgand22
ahgand22 74065 74002  0 21:21 ?        00:00:00 sshd: ahgand22@notty
ahgand22 74067 74065  0 21:21 ?        00:00:00 bash
```

G. Yes, the child does get a new parent. The child is adopted by init and its ppid will be 1.



```
ahgand22@radius:~/csci346/project1$ ./launcher 1 2 2 2 3
I am Process Eve. I am the parent. My Id is 58162
Eve (Process 58162, parent 29455 ) will sleep for 3.000000 seconds
Cal (process 58165, parent 58163) with 2 rolls left, rolls 19 and now has 22 points.
Ada (process 58163, parent 58162) with 1 rolls left, rolls 9 and now has 12 points.
Bev (process 58164, parent 58162) with 2 rolls left, rolls 1 and now has 4 points.
Dot (process 58166, parent 58163) with 2 rolls left, rolls 18 and now has 21 points.
Cal (process 58165, parent 58163) with 1 rolls left, rolls 19 and now has 41 points.
Ada (process 58163, parent 58162 finished in 1.000233303 seconds, quits with exit code 12)
Bev (process 58164, parent 58162) with 1 rolls left, rolls 10 and now has 14 points.
Dot (process 58166, parent 1) with 1 rolls left, rolls 17 and now has 38 points.
Cal (process 58165, parent 1 finished in 2.000377912 seconds, quits with exit code 41)
Bev (process 58164, parent 58162 finished in 2.000358985 seconds, quits with exit code 14)
Dot (process 58166, parent 1 finished in 2.000764871 seconds, quits with exit code 38)
Eve (Process 58162, parent 29455) will now call wait().
Eve (Process 58162, parent 29455) got pid 58163 and status 12 from wait.
```

H. Siblings of a killed process continue running normally.



```
ahgand22@radius:~/csci346/project1$ ./launcher 10 10 2 2 12
I am Process Eve. I am the parent. My Id is 61436
Eve (Process 61436, parent 29455 ) will sleep for 12.000000 seconds
Cal (process 61439, parent 61437) with 2 rolls left, rolls 3 and now has 6 points.
Ada (process 61437, parent 61436) with 10 rolls left, rolls 16 and now has 19 points.
Bev (process 61438, parent 61436) with 10 rolls left, rolls 10 and now has 13 points.
Dot (process 61440, parent 61437) with 2 rolls left, rolls 8 and now has 11 points.
Cal (process 61439, parent 61437) with 1 rolls left, rolls 19 and now has 25 points.
Ada (process 61437, parent 61436) with 9 rolls left, rolls 1 and now has 20 points.
Bev (process 61438, parent 61436) with 9 rolls left, rolls 13 and now has 26 points.
Dot (process 61440, parent 61437) with 1 rolls left, rolls 17 and now has 28 points.
Cal (process 61439, parent 61437 finished in 2.000373904 seconds, quits with exit code 25)
Ada (process 61437, parent 61436) with 8 rolls left, rolls 16 and now has 36 points.
Bev (process 61438, parent 61436) with 8 rolls left, rolls 20 and now has 46 points.
Dot (process 61440, parent 61437 finished in 2.001109539 seconds, quits with exit code 28)
Ada (process 61437, parent 61436) with 7 rolls left, rolls 14 and now has 50 points.
Bev (process 61438, parent 61436) with 7 rolls left, rolls 7 and now has 5 points.
Ada (process 61437, parent 61436) with 6 rolls left, rolls 16 and now has 6 points.
Bev (process 61438, parent 61436) with 6 rolls left, rolls 16 and now has 21 points.
Ada (process 61437, parent 61436) with 5 rolls left, rolls 13 and now has 19 points.
Bev (process 61438, parent 61436) with 5 rolls left, rolls 1 and now has 22 points.
Ada (process 61437, parent 61436) with 4 rolls left, rolls 16 and now has 35 points.
Bev (process 61438, parent 61436) with 4 rolls left, rolls 10 and now has 32 points.
Ada (process 61437, parent 61436) with 3 rolls left, rolls 6 and now has 41 points.
Bev (process 61438, parent 61436) with 3 rolls left, rolls 15 and now has 47 points.
Ada (process 61437, parent 61436) with 2 rolls left, rolls 8 and now has 49 points.
Ada (process 61437, parent 61436) with 1 rolls left, rolls 20 and now has 6 points.
Ada (process 61437, parent 61436 finished in 10.001524443 seconds, quits with exit code 6)
Eve (Process 61436, parent 29455) will now call wait().
Eve (Process 61436, parent 29455) got pid 61437 and status 6 from wait.
```

Experiments 3

A. The bash terminal where cointoss program was running was killed. The reason behind this is that parent of cointoss is the bash program itself. As described in class, bash is just a program, and thereby killing it will stop the program and close/kill the bash (parent).

B. I have added some code that determines the signal number (if killed by signal), and if killed because of core dump.