Lab 3

Thursday, 22 August 2024 11:36 AM

3.1 Task 1: Understanding the process tree (20%)

Try tracing the process tree for a few other processes other than `bash`, in the `ptree.txt` file you created in Section 2.2. What can you observe?

Now, try running the `htop` command as follows:

```
$ htop -t
```

Is the result of running `htop` in agreement with your results from manually tracing the process tree earlier?

Submit the `ptree.txt` file that you have created earlier, together with a brief summary of your findings.

Running the htop -t command displays a lot more processes compared to the ptree.txt file, this is because it is displaying all the child processes too whereas ps only displays a selection of the active processes, htop also continuously updates what processes are being run.

Run the program, and look at the output and trace back the sequence of parent process IDs for the child process. **Document your findings for the following**:

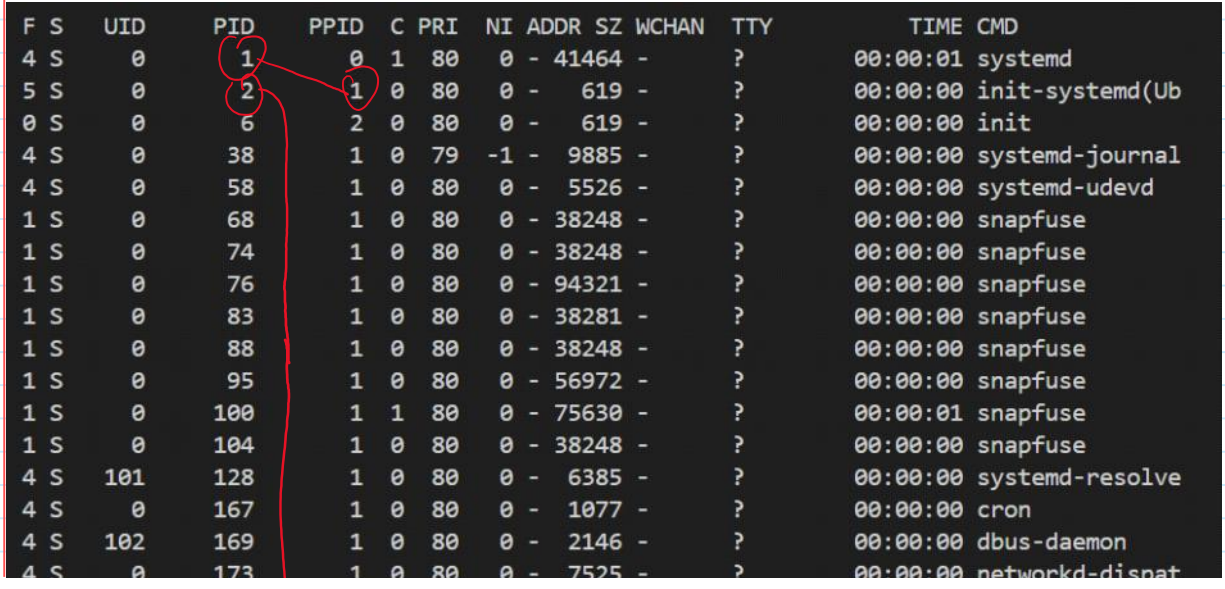
(a) Locate the child process in the listing and underline its `PID`;

(b) Put a circle around the parent's process ID for that child (it's right next to the `PID`);

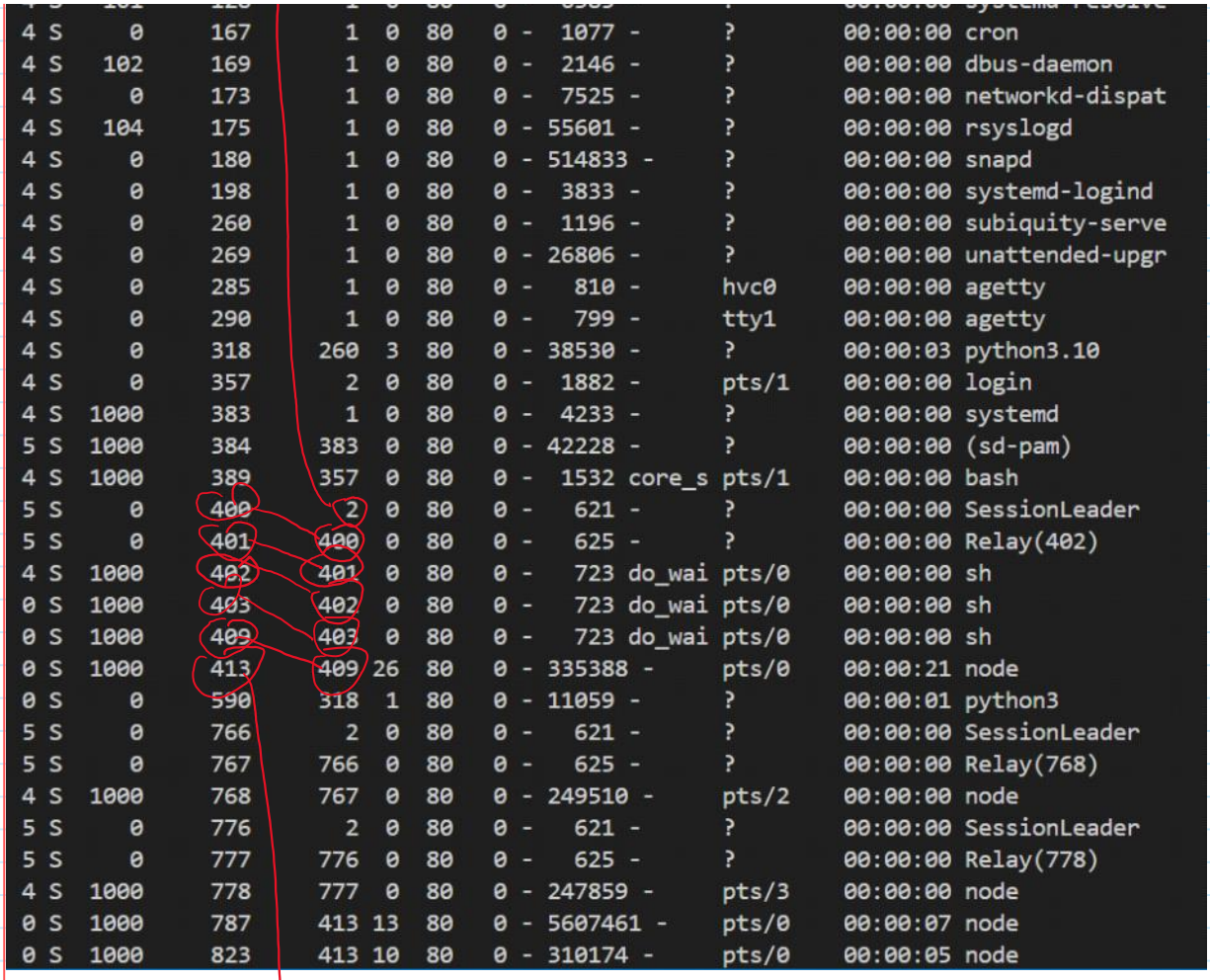
(c) Draw a line from the circled parent's ID to the PID for that process (find it on a previous line);

(d) Repeat this for the parent, and its parent, etc., until you reach the process with `PID 1`.

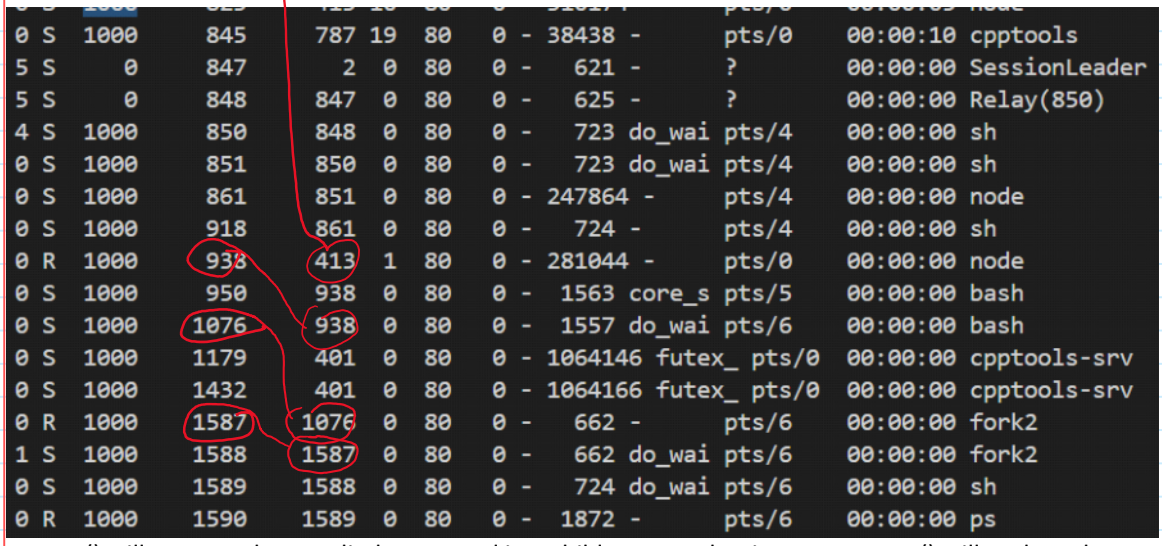
In addition to your findings above, are there any differences between using `exec()` and `system()`? If there are differences, can you list two of them?



F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
4	S	0	1	0	1	80	0	-	41464	-	?	00:00:01	systemd
5	S	0	2	1	0	80	0	-	619	-	?	00:00:00	init-systemd(Ub
0	S	0	6	2	0	80	0	-	619	-	?	00:00:00	init
4	S	0	38	1	0	79	-1	-	9885	-	?	00:00:00	systemd-journal
4	S	0	58	1	0	80	0	-	5526	-	?	00:00:00	systemd-udevd
1	S	0	68	1	0	80	0	-	38248	-	?	00:00:00	snapfuse
1	S	0	74	1	0	80	0	-	38248	-	?	00:00:00	snapfuse
1	S	0	76	1	0	80	0	-	94321	-	?	00:00:00	snapfuse
1	S	0	83	1	0	80	0	-	38281	-	?	00:00:00	snapfuse
1	S	0	88	1	0	80	0	-	38248	-	?	00:00:00	snapfuse
1	S	0	95	1	0	80	0	-	56972	-	?	00:00:00	snapfuse
1	S	0	100	1	1	80	0	-	75630	-	?	00:00:01	snapfuse
1	S	0	104	1	0	80	0	-	38248	-	?	00:00:00	snapfuse
4	S	101	128	1	0	80	0	-	6385	-	?	00:00:00	systemd-resolve
4	S	0	167	1	0	80	0	-	1077	-	?	00:00:00	cron
4	S	102	169	1	0	80	0	-	2146	-	?	00:00:00	dbus-daemon
4	S	0	173	1	0	80	0	-	7525	-	?	00:00:00	networkd-dispat



```
4 S     0     167       1  0  80   0 -  1077 -      ?        00:00:00 cron
4 S   102     169       1  0  80   0 -  2146 -      ?        00:00:00 dbus-daemon
4 S     0     173       1  0  80   0 -  7525 -      ?        00:00:00 networkd-dispat
4 S   104     175       1  0  80   0 - 55601 -      ?        00:00:00 rsyslogd
4 S     0     180       1  0  80   0 - 514833 -     ?        00:00:00 snapd
4 S     0     198       1  0  80   0 -  3833 -      ?        00:00:00 systemd-logind
4 S     0     260       1  0  80   0 -  1196 -      ?        00:00:00 subiquity-serve
4 S     0     269       1  0  80   0 - 26806 -      ?        00:00:00 unattended-upgr
4 S     0     285       1  0  80   0 -   810 -      hvc0     00:00:00 agetty
4 S     0     290       1  0  80   0 -   799 -      tty1     00:00:00 agetty
4 S     0     318     260  3  80   0 - 38530 -      ?        00:00:03 python3.10
4 S     0     357       2  0  80   0 -  1882 -      pts/1    00:00:00 login
4 S  1000     383       1  0  80   0 -  4233 -      ?        00:00:00 systemd
5 S  1000     384     383  0  80   0 - 42228 -      ?        00:00:00 (sd-pam)
4 S  1000     389     357  0  80   0 -  1532 core_s pts/1    00:00:00 bash
5 S     0     400       2  0  80   0 -   621 -      ?        00:00:00 SessionLeader
5 S     0     401     400  0  80   0 -   625 -      ?        00:00:00 Relay(402)
4 S  1000     402     401  0  80   0 -   723 do_wai pts/0    00:00:00 sh
0 S  1000     403     402  0  80   0 -   723 do_wai pts/0    00:00:00 sh
0 S  1000     409     403  0  80   0 -   723 do_wai pts/0    00:00:00 sh
0 S  1000     413     409 26  80   0 - 335388 -     pts/0    00:00:21 node
0 S     0     590     318  1  80   0 - 11059 -      ?        00:00:01 python3
5 S     0     766       2  0  80   0 -   621 -      ?        00:00:00 SessionLeader
5 S     0     767     766  0  80   0 -   625 -      ?        00:00:00 Relay(768)
4 S  1000     768     767  0  80   0 - 249510 -     pts/2    00:00:00 node
5 S     0     776       2  0  80   0 -   621 -      ?        00:00:00 SessionLeader
5 S     0     777     776  0  80   0 -   625 -      ?        00:00:00 Relay(778)
4 S  1000     778     777  0  80   0 - 247859 -     pts/3    00:00:00 node
0 S  1000     787     413 13  80   0 - 5607461 -    pts/0    00:00:07 node
0 S  1000     823     413 10  80   0 - 310174 -     pts/0    00:00:05 node
```



```
0 S  1000     845     787 19  80   0 - 38438 -      pts/0    00:00:10 cpptools
5 S     0     847       2  0  80   0 -   621 -      ?        00:00:00 SessionLeader
5 S     0     848     847  0  80   0 -   625 -      ?        00:00:00 Relay(850)
4 S  1000     850     848  0  80   0 -   723 do_wai pts/4    00:00:00 sh
0 S  1000     851     850  0  80   0 -   723 do_wai pts/4    00:00:00 sh
0 S  1000     861     851  0  80   0 - 247864 -     pts/4    00:00:00 node
0 S  1000     918     861  0  80   0 -   724 -      pts/4    00:00:00 sh
0 R  1000     938     413  1  80   0 - 281044 -     pts/0    00:00:00 node
0 S  1000     950     938  0  80   0 -  1563 core_s pts/5    00:00:00 bash
0 S  1000    1076     938  0  80   0 -  1557 do_wai pts/6    00:00:00 bash
0 S  1000    1179     401  0  80   0 - 1064146 futex_ pts/0  00:00:00 cpptools-srv
0 S  1000    1432     401  0  80   0 - 1064166 futex_ pts/0  00:00:00 cpptools-srv
0 R  1000    1587    1076  0  80   0 -   662 -      pts/6    00:00:00 fork2
1 S  1000    1588    1587  0  80   0 -   662 do_wai pts/6    00:00:00 fork2
0 S  1000    1589    1588  0  80   0 -   724 do_wai pts/6    00:00:00 sh
0 R  1000    1590    1589  0  80   0 -  1872 -      pts/6    00:00:00 ps
```

system() will execute the supplied command in a child process that it spawns. exec() will replace the current process with the invocation of the new executable that you specify

3.3 Task 3: Understanding process scheduling (30%)

Refer to the program (`fork.c`) which you have downloaded in the pre-class preparation, Section 2.3. Notice the `for-loop` with the loop variable '`j`'. This is a *delay* loop, to make the parent and child processes take a longer time to complete their work. If you haven't already done so, try running the `fork` program several times. **What can you say about the output?**

The order of the child and parent changes in the latter iterations is different each run of the program

Now, to see the effect of the delay loop, change the limit value by dropping one zero (to make it 100,000). Re-compile and run the program. **What has happened to the pattern of executions of the two processes?** Continue to drop one zero from the limit value until the value of the limit is 100. **Are there any changes?**

The lower the limit the more child processes are at the end , when the limit is high it alternates between parent and child processes but as the limit is lower the parent processes will finish earlier and the child processes will be left at the end

Now, what if we added more zeroes to initial limit value of 1,000,000 instead, so that the value of the limit is 10,000,000, or 100,000,000? **Does this result in anything different?**

It takes a lot longer for the program to finish running

By the way, there is an alternative (less precise but simpler) method for waiting — but in integer chunks of *seconds*. Now, change the waiting code in the program with the following bit of code:

```
1  sleep(2);
```

Re-compile and run the program again. **What differences in the output do you notice?**

Yes a pair of child and parent processes run and finish and then print rather than all of the processes printing at the same time