CS-323 Exercises Solutions Week 2

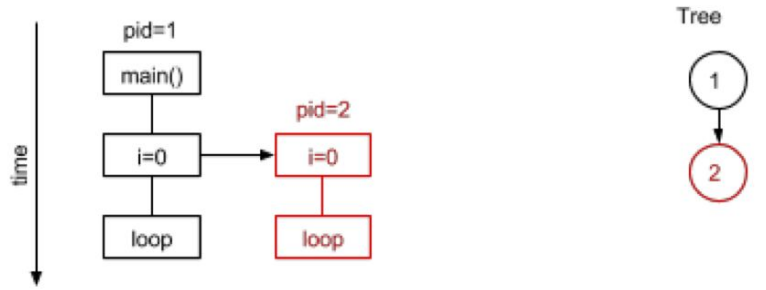
01 Mar 2019

1) Process trees

a)

Step	Command	Process Tree
1.		csh (initial process)
2.	fork	csh – csh
3.	exec	csh – make
4.	fork	csh – make – make
5.	exec	csh – make – foo
6.	fork	csh – make – foo \ make
7.	exec	csh – make – foo \ bar
8.	exit	csh – make – bar or foo (make and bar can also finish before foo)
9.	exit	csh – make
10.	exit	csh

b)



Explanations:

- one color = one process
- the arrow means fork()ing, in the direction of the child
- we show an idealized timeline; in reality the processes will be scheduled at different times
- because child pids depend on the scheduling order, in reality processes may have different pids than shown

2) Linux fork

a)

0112222 or 0121222 or 0122122

Depending on the order in which children get executed.

b)

00000000000000000000...
The program will never terminate. Every time the execution reaches execl, the same program gets started from the beginning.

3) I/O Redirection under the hood

Question 1:

The program prints:

aaaaa
bbbbb

After executing dup(i), both i and j point to the same entry in the global file table and thus share the same byte offset in the file. Thus, since the next two read operations read different parts of the file, buf1 and buf2 contain different data.

Question 2:

The program prints:

aaaaa
bbbbb

After fork() the child inherits copies of the parent's set of open file descriptors. Each file descriptor in the child refers to the same open file description (see open(2)) as the corresponding file descriptor in the parent. This means that the two file descriptors share open file status flags, file offset, etc.

Question 2:

```
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>

int main(void)
{
      char *argv[] = { "/bin/ls", "-la", 0 };
      char *envp[] =
      {
            "HOME=/",
            "PATH=/bin:/usr/bin",
            0
      };
      int fd = open("output.txt", O_CREAT|O_WRONLY);

      //dup2(fd, 1); // comment in this line and comment out the next 3
       close(1);
      dup(fd);
      close(fd);

      // execute ls command, with redirected output
      execve(argv[0], &argv[0], envp);
```

```
	return -1; //something went wrong
}
```

4) Linux pipes

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include<sys/wait.h>

int main(int argc, char* argv[])
{
	int pipefds[2], i;
	pid_t pid;
	char buf[30];

	//create pipe
	if(pipe(pipefds) == -1) {
		perror("pipe");
		exit(EXIT_FAILURE);
	}

	memset(buf,0,30);
	pid = fork();
	if (pid > 0) {
		printf("PARENT write in pipe\n");
		//parent close the read end
		close(pipefds[0]);
		//parent write in the pipe write end
		write(pipefds[1], "CS323", 6);
		//after finishing writing, parent close the write end
		close(pipefds[1]);
		//parent wait for child
		wait(NULL);
	} else {
		//child close the write end
		close(pipefds[1]);
		//child read from the pipe read end until the pipe is empty
		i = 0;
		while(read(pipefds[0], &buf[i++], 1) == 1);
		printf("CHILD read from pipe -- %s\n", buf);
		//after finishing reading, child close the read end
		close(pipefds[0]);
		printf("CHILD: EXITING!\n");
		exit(EXIT_SUCCESS);
	}
	return 0;
}
```