Part 1

1.

```
1
2   // question 1
3   memset(&pcb, 0, sizeof pcb);
4
```

2.

```
4
5   // question 2
6   if (buf[strlen(buf) - 1] == '\n'){
7       buf[strlen(buf) - 1] = 0;
8   }
9
```

3. The function strtok takes 2 arguments, the first one is the string that we wanna process and the second one is the C string containing the delimiters. We would call strtok(inStr, WSPACE) to get the token and use strtok(NULL, WSPACE) to get the other tokens in the string during the while loop. And the count will be in the while loop that stops until it is null. We will use strcpy to copy the token into the list. We will use an array of strings to store the obtained tokens.

```
// question 3
char *token;
char strs[MAXLINE][MAXWORD];
int count = 0;
   // get the first token
   token = strtok(inStr, WSPACE);
   // get all tokens
   while( token != NULL ) {
    strcpy(strs[count], token)  // store it in a list
    count =  count + 1; // count the numbers
    token = strtok(NULL, WSPACE); // in order to get next token
    // and to continue with the same string
   }
```

4.

```
// question 4
fprintf(stdout, "%s\n", strerror(errno));
```

Part 2

Experiment 1 Command-line: “alp2 w”

	PID	PPID	State	CMD
1. (a1p2 process)	1875284	1809988	S+	a1p2
2. (child process)	1875285	1875284	S+	myclock
3. (grand child (if any))	1875531	1875285	S+	sleep
4. (a1p2 process)				
5. (child process)	1875285	1	S	myclock
6. (grand child (if any))	1877184	1875285	S	sleep

From the ids, we can see 1875531(sleep) is a child of 1875285(myclock) and is a child of 1875284(a1p2) and it's a child of 1809988 through the ppid of each. All of them are in the S+ state which means they are in interruptible sleep and waiting for an event to complete but also part of the foreground process group. After terminating the a1p2 process. We see the parent of myclock is changed to 1 and sleep is a different sleep. Both of the two are in interruptible sleep.

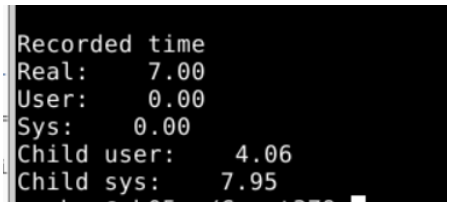
Experiment 2: Command-line: “alp2 s”

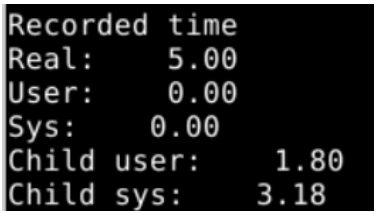
	PID	PPID	State	CMD
1. (a1p2 process)	1884186	1809988	S+	a1p2
2. (child process)	1884187	1884186	S+	mylock
3. (grand child (if any))	1884226	1884187	S+	sleep
4. (a1p2 process)	1884186	1809988	S+	a1p2
5. (child process)	1884187	1884186	Z+	myclock <defunct>
6. (grand child (if any))				

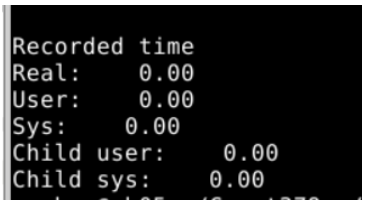
From the ids, we can see 1884226(sleep) is a child of 1884187(myclock) and is a child of 1884186(a1p2) and it's a child of 1809988 through the ppid of each. All of them are in the S+ state which means they are in interruptible sleep and waiting for an event to complete but also part of the foreground process group. After terminating the myclock process. We see that there is no longer a grandchild and myclock is in Z+ state which is the zombie state and terminated but not reaped by its parent. And a1p2 is still in S+ state which is in interruptible sleep and waiting for an event to complete.

Part 3

Running testfile.data
The time recorded are for the choices -1, 1 and 0, and in that order are the photos.







When running the test case and choosing -1, we can see that the child system time is higher than the real time with 7.95s and the real time is 7 seconds. And both of the user and sys time is 0.00, as a1p3 only creates the child process that runs the commands and it does not run the commands itself. It is reasonable as it represents actual elapsed time, while the child user and child sys values represent CPU execution time of the child process. When choosing 1 and running the test case, we see our real time is 5 s and is lower than the time of choosing -1. This is because the program just waits for one of the child processes to finish, and the fastest one to finish happens to be that amount of real time. And the child cpu time is 3.18 that is lower than 7.95 because we are only showing the time after finishing one child process. When choosing 0 and running the test we see the running times are all very small numbers such that it displays 0.00. This is because the a1p3 did not wait for any of its children to process, and it can finish in a very short time as it does not run these commands as the child processes do. And since a1p3 finishes, the child user and sys will show 0.00 as the print of the time is in a1p3.

And these are the commands:

```
print_cmd(): [timeout 7 ./myclock outA]
print_cmd(): [timeout 7 ./myclock outB]
print_cmd(): [timeout 7 ./myclock outB]
print_cmd(): [timeout 5 ./do_work bash.man 100]
print_cmd(): [timeout 7 ./do_work bash.man 50]
```

Acknowledgments:
Advanced Programming in the UNIX® Environment 3rd edition