3.8

Short-term: this selects a process from the processes currently ready to execute, the difference between short and long term scheduling is how much more frequently short-term scheduling occurs than long-term (often once ever 100ms)
Medium-term: in time-sharing systems this allows for swapping to occur, or the removal of a process from memory only to reintroduce it later to streamline execution, useful in cases where processes may be attempting to access more memory than is available and so some memory needs to be freed up
Long-term: this scheduler selects processes from a mass-storage device and loads them into memory

3.9

It saves the state of the currently running process to a process control block, then loads the state of the next process from the process control block.

3.12

Number of processes:
Parent Process = 1
# of children = 1 per loop iteration (4)

Total number of processes (including parent) is five.

3.14

A – 0 (currently zero because of fork())
B – 2603 (child pid)
C – 2603 (child pid)
D – 2600 (parent pid)

The following is sample code written to show this (copied example)

```
Anon-2:test masonschreck$ ./a.out
C --  79803:
D -- 79802:
A -- 0:
B --  79803:
Anon-2:test masonschreck$ vi test.c
Anon-2:test masonschreck$
```

```c
#include <stdio.h>
int main()
{
    int pid, pid1;
    pid = fork();
    if(pid<0){
        printf("oops");
        return 1;
    }
    else if(pid==0){
        //child one
        pid1 = getpid();
        printf("A -- %d:\n",pid);
        printf("B --  %d:\n",pid1);
    }
    else{
        pid1 = getpid();
        printf("C --  %d:\n",pid);
        printf("D -- %d:\n",pid1);
        wait(NULL);
    }
return 0;
}
```

3.17

The output at line X will be CHILD: followed by the negative square of 0 through 4.
(e.g. – CHILD: 0
CHILD: -1
CHILD: -4
CHILD: -9
CHILD: -16
)

The output at line Y will simply output the following PARENT: (0-4).
e.g. –
PARENT 0
PARENT 1
PARENT 2
PARENT 3
PARENT 4