Homework #4
Due We. 11/14

This homework can be done in a group of maximum two students. Please prepare a report including the source code, captures of output and send it along with related source files as a zip file to my blackboard. Please write the name of your teammate in your report. The zip file should be named by your family name. Please print the report and bring the hard copy by the due date to the class. The selective grading will be used for homeworks of this course.

1. Write out the memory map for the following code, providing all values at the end of execution. It can be written using multiple maps, or areas of memory, one for each process. What is the exact output produced by this program?

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

main()
{
int     i,j,k;

k=0;
for (j=0; j<4; j++)
  k=k+j;
i=fork();
if (i == 0)
  for (i=3; i<k; i++)
    j=j-i;
else
  i=k%3;
printf("%d %d %d\n",i,j,k);
}
```

2. Create a library. The library should contain two functions, Hello() and World(). Calling the Hello() function should print out the string "Hello"; calling the World() function should print out the string "World." The library should use a flag (implemented using a static global variable) that keeps track of which string was last printed. A call to the Hello() function should print out "Hello" only if "World" was the last string printed, and vice versa. The flag should be updated at the end of each function call. Each function should return 1 if the string was printed, 0 otherwise. Write a simple program that uses the library and tests for the correct working of the functions.

3. The Unix *mkdir* command accepts a *-p* option. The *-p* option will be used if you are trying to create a directory with top-down approach (e.g., creating a directory with absolute path). That's create parent directory then child and so on if none exists. Write a version of *mkdir* that supports this option.

4. Expand the code in *waitdemo1.c* so that the parent creates two children and then waits until both children exit. Expand your solution further so the program accepts an integer as a command-line argument. The program then creates the number of child processes and assigns each child a random number of seconds to sleep. Finally, the parent process reports as each child exits.

5. What is the exact output of the following code?

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

main()
{
int     i,j;

i=fork();
for (j=0; j<3; j++)
  {
  if (i == 0  &&  j == 0)
    {
    sleep(3);
    printf("Cats\n");
    }
  else if (i == 0)
    {
    sleep(2);
    printf("Dogs\n");
    }
  else
    {
    sleep(2);
    printf("Raining\n");
    }
  }
}
```

6. What is the functionality of the following code?

```
#include <stdio.h>
#include <unistd.h>
main()
{
int     i,j;

j=0;
printf("Ready to fork...\n");
i=fork();
if (i == 0)
  {
  printf("The child executes this code.\n");
  for (i=0; i<5; i++)
    j=j+i;
  printf("Child j=%d\n",j);
  }
else
  {
  j=wait();
  printf("The parent executes this code.\n");
  printf("Parent j=%d\n",j);
  }
}
```