Lab # 8

Instructor: Dr. Raheel A. Memon

Objectives:

- Implicit threading using openmp
- Pipes

Demo – 1:

Demonstrate number of cores with openmp:

```
#include <omp.h>
#include <stdio.h>
int main(int argc, char *argv[])
{
      /* sequential code */
      #pragma omp parallel
      {
      printf("I am a parallel region.");
      }
      /* sequential code */
      return 0;
}
```

To execute with openmp use –fopenmp **while using gcc compiler, on intel compiler use** -openmp

Demo – 2:

```
#include <omp.h>

#include <stdio.h>

int main(int argc, char *argv[])

{

/* sequential code */

#pragma omp parallel    // compiler directive

{

printf("I am a parallel region.\n");
```

```
}
/* sequential code */
return 0;
}
```

For gcc compiler add –fopenmp

For Intel compiler add –openmp

```
$gcc simple.c –o simple –fopenmp
$ ./simple
```

I am a parallel region.

I am a parallel region.

Specify the number of threads for openmp in environment variables

```
$ export OMP_NUM_THREADS=4
$ ./simple
```

I am a parallel region.

I am a parallel region.

I am a parallel region.

I am a parallel region.

Task # 1

Run for loop with openmp to implement following in parallel:

$$\boldsymbol{Sum} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{i}$$

Task # 2

Parallel function execution:

Implement the myturn and yourturn example with openmp.

Task # 3

Now get the thread id of running threads for myturn and yourturn using openmp.

Pipes

Example # 1: Pipe Function

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void)
{
        int     fd[2], nbytes;
        pid_t   childpid;
        char    string[] = "Hello, world!\n";
        char    readbuffer[80];
        pipe(fd);

        if((childpid = fork()) == -1)
        {
                perror("fork");
                exit(1);
        }
        if(childpid == 0)
        {
                /* Child process closes up input side of pipe */
                close(fd[0]);

                /* Send "string" through the output side of pipe */
                write(fd[1], string, (strlen(string)+1));
                exit(0);
        }
        else
```

```
        {
                /* Parent process closes up output side of pipe */
                close(fd[1]);

                /* Read in a string from the pipe */
                nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
                printf("Received string: %s", readbuffer);
        }
        return(0);
}
```

Task #4:

Implementation of pipes

Write a program, where parent declares a number, which is passed to child process. Child process then multiplies that number with let's say 4 and returns the result to parent process.

Task #5:

Implementation of named pipes. Refer to book and implement task # 3 with named pipes