CENG328
Operating Systems

Laboratory VI
Processes II & Threads I

1. Processes

- **execl** family of functions; **execl execlp execle exect execv execvp** all performs a similar function by starting another program. This new program overlays the existing program, so you can never return to the to original code unless the call to execl fails.

code21.c:

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

main()
{
    int pid;
    if ((pid = fork()) == 0) {
        execl("/bin/ls", "ls", "/", 0);
    }
    else {
        wait(&pid);
    printf("Exec finished\n");
    }
}
```

code22.c:

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

int main()
{
    printf("Running ps with execlp\n");
    execlp("ps", "ps", "-ax", 0);
    printf("Done.\n");
    exit(0);
}
```

2. Threads

- **Important:** You should compile the codes with **-lpthread** parameter, such as:

  $ gcc -o thread thread.c -lpthread

- Creating a (child) thread; **pthread_t**, **pthread_create**. code23.c
  - You should break with **CTRL + C**.
  - Study the arguments in the function; pthread create.
  - Execute several times. Does the thread IDs change in number? Is this what you expected?

- Creating a child thread (with error checking) and joining (existing a thread); **pthread_join**. code24.c
  - You should supply a message by means of **argv**.
  - Study the arguments in the functions; **pthread_create**, **pthread_join**.
  - Execute several times. Does the thread IDs change in number? Is this what you expected?
  - What is the functionality of joining the thread?

2. Threads

- Competing; - code25.c
  - You should break with **CTRL + C**.
  - Execute this code several times. You should observe that the pattern for the print messages of the main and the new change. Why?
- The parent and one child; same operations with different data. code26.c
- The parent and four childrens; same operations with same data. code27.c
  - Notice that **x** is global data.
  - We use the same function for each thread.
  - Execute this code several times. Do you get the same order for thread execution? Is that what you expect?
  - What would change if we had more than one function for each thread?

3. Exercises

- Complete the following program (..... are for the missing parts) that creates two threads and each print different characters in given amount.

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

/* Parameters to print_function. */
struct char_print_parms
{
    /* The character to print. */
    char character;
    /* The number of times to print it. */
    int count;
};

/* Prints a number of characters to stderr, as given by PARAMETERS,
which is a pointer to a struct char_print_parms. */
void* char_print (void* parameters)
{
    /* Cast the cookie pointer to the right type. */
    struct char_print_parms* p = (struct char_print_parms*) parameters;
    int i;
    for (i = 0; i < p->count; ++i)
        fputc (p->character, stderr);
    return NULL;
}
// ...
```

3. Exercises

```
// ...
int main ()
{
    pthread_t thread1_id;
    .......

    struct char_print_parms thread1_args;
    .......

    /* Create a new thread to print 30000 x's. */
    thread1_args.character = 'x';
    thread1_args.count = 30000;
    .......

    /* Create a new thread to print 20000 o's. */
    thread2_args.character = 'o';
    .......
    .......

    /* Make sure the first thread has finished. */
    pthread_join (thread1_id, NULL);

    /* Make sure the second thread has finished. */
    .......

    /* Now we can safely return. */
    return 0;
}
```

3. Exercises

- Write a program to calculate the factorial of a number. Your program should;
  - create a thread,
  - supply a function for the thread,
  - output the result in the main (function/process/thread).