1 OPERATING SYSTEMS LABORATORY V - Processes II& Threads I

1.1 Examples&Exercises:

- Compile and run the code.
- Analyze the code and output.
- You should compile with **-lpthread** whenever necessary, such as

```
$ gcc -o thread thread.c -lpthread
```

1. **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 and code22.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");
  }
}
```

```
#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. 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?
3. 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?

4. 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?
5. The parent and one child; same operations with *different* data. code26.c
6. 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?
7. 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;
}
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;
}
```

1.2 Assignment II

In this assignment, you're to sort by creating processes.

- one parent and two childs,
  - parent is responsible for creating the numbers (by any means; random generator, reading from a file unsorted.txt), ...),
  - after generating, put to an array, creating first child,
  - the first child should create the second one,
  - each child should sort the half of the array.
- use the sorting algorithm you used in the previous assignment,
- do the error checking whenever it is necessary (if array is allocated or if childs are created or if file is opened succesfully, etc.),
- make sure that parent does not exit before childs complete.