**CISC2005 Exercise 04**

# Prerequisites

## Successful installation of \*nix terminal (e.g., WSL, cygwin) or a \*nix operating system (e.g., Linux, Mac).

For Windows system, I recommend you to install WSL. <https://learn.microsoft.com/en-us/windows/wsl/install>

## Successful installation of a C compiler, GCC is recommended.

## Successful installation of a text editor, VIM or Emacs is recommended.

# Answering Questions raised in last week

1. In last week, someone noticed that some codes may sometimes behave strangely, i.e., some output repeated twice and the *printf* function seems like being executed repeatedly (typically in eclipse + cygwin environment). The reason is the buffering mechanism of *stdout* . Specifically, once you observe a repeated printing, you could add a *sleep(5)* command before *fork()* , and then you can see that before sleeping the desired output is not yet printed. After the *fork()* is called, two processes are simultaneously running so the buffered outputs are printed twice. To mitigate this behaviour, you could palce a *fflush(stdout)* command before calling the *fork()* function, enforcing flush of *stdout* in advance.

# Compile Instruction for thread-based program

In case of using Linux system, you need to link with *pthread* library using *-l pthread* during compile phase. Otherwise, you will fail to compile the source code. As an example, you should change your compile command from *gcc q1.c -o q1* to *gcc q1.c -l pthread -o q1*.

Eclipse or other IDE would automatically assemble this library during compiling phase so you don’t have to explicitly do so.

# Tasks

In this section, students are required to execute the following codes, and try to understand the behaviour. For each program, students should capture a screenshot of the successful execution, and answer attached questions briefly. In the submission file, please attach the **execution screenshot** and the **explanation of each question** in sequence.

Q1: Execute the following code, and state why two PIDs are the same and why two threadIDs are different.

*#include <stdio.h>*

*#include <unistd.h>*

*#include <pthread.h>*

*void kidfunc() {*

*printf("Kid PID is %d, thread ID is %d\n", getpid(), pthread\_self());*

*}*

*int main() {*

*pthread\_t kid;*

*pthread\_create(&kid, NULL, kidfunc, NULL);*

*printf("Parent PID is %d, thread ID is %d\n", getpid(), pthread\_self());*

*pthread\_join(kid, NULL);*

*printf("Complete!\n");*

*return 0;*

*}*

Q2: Execute the following code, and state if the global data in the last output is deterministic or not and why.

*#include <stdio.h>*

*#include <unistd.h>*

*#include <pthread.h>*

*int glob\_data = 5;*

*void\* kidfunc() {*

*glob\_data = 15;*

*}*

*int main() {*

*pthread\_t kid;*

*printf("Start of program. Global data = %d.\n", glob\_data);*

*pthread\_create(&kid, NULL, kidfunc, NULL);*

*glob\_data = 10;*

*pthread\_join(kid, NULL);*

*printf("End of program. Global data = %d.\n", glob\_data);*

*return 0;*

*}*

Q3: Execute the following code, and state 1) why the variable *this\_is\_global* equals to 1002 in thread test, 2) Identify whether the address of variables in two threads are identical in thread test and why 3) why the modification occurred in child process doesn’t influence the value in parent process, 4) why the address of variables in two processes are identical in process test.

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <pthread.h>*

*#include <unistd.h>*

*int this\_is\_global;*

*void thread\_func(void \*dummy) {*

*int local\_thread;*

*printf("Thread %d, pid %d, addresses: &this\_is\_global: %X, &local: %X\n",*

*pthread\_self(), getpid(), &this\_is\_global, &local\_thread);*

*this\_is\_global++;*

*printf("In Thread %d, incremented this\_is\_global=%d\n", pthread\_self(),*

*this\_is\_global);*

*pthread\_exit(0);*

*}*

*int main() {*

*// thread test*

*pthread\_t thread1, thread2;*

*printf("First, we create two threads to see better what context they share...\n");*

*this\_is\_global = 1000;*

*printf("Set this\_is\_global=%d\n", this\_is\_global);*

*pthread\_create(&thread1, NULL, (void\*) &thread\_func, (void\*) NULL);*

*pthread\_create(&thread2, NULL, (void\*) &thread\_func, (void\*) NULL);*

*pthread\_join(thread1, NULL);*

*pthread\_join(thread2, NULL);*

*printf("After threads, this\_is\_global=%d\n", this\_is\_global);*

*printf("\n");*

*printf("After testing threads, let's call fork..\n");*

*// process test*

*int local\_main = 17;*

*this\_is\_global = 17;*

*printf("Before fork(), local\_main=%d, this\_is\_global=%d\n", local\_main, this\_is\_global);*

*fflush(stdout);*

*int pid = fork();*

*if (pid == 0) {*

*printf("In child, pid %d: &this\_is\_global: %X, &local\_main: %X\n", getpid(), &this\_is\_global, &local\_main);*

*local\_main = 13;*

*this\_is\_global = 23;*

*printf("In child, local main=%d, this\_is\_global=%d\n", local\_main, this\_is\_global);*

*exit(0);*

*} else {*

*printf("In parent, pid %d: &this\_is\_global: %X, &local\_main: %X\n", getpid(), &this\_is\_global, &local\_main);*

*wait(NULL);*

*printf("In parent, local\_main=%d, this\_is\_global=%d\n", local\_main, this\_is\_global);*

*exit(0);*

*}*

*}*

Q4: Execute the following code multiple times and state why the result is not deterministic.

*#include <stdio.h>*

*#include <pthread.h>*

*#include <stdlib.h>*

*#include <unistd.h>*

*int tot\_items = 0;*

*struct kidrec {*

*int data;*

*pthread\_t id;*

*};*

*#define NKIDS 10*

*void\* kidfunc(void \*p) {*

*int \*ip = (int\*) p;*

*int tmp, n;*

*tmp = tot\_items;*

*for (n = 500000; n--;)*

*tot\_items = tmp + \*ip;*

*}*

*int main() {*

*struct kidrec kids[NKIDS];*

*for (int m = 0; m < NKIDS; ++m) {*

*kids[m].data = m + 1;*

*pthread\_create(&kids[m].id, NULL, kidfunc, &kids[m].data);*

*}*

*for (int m = 0; m < NKIDS; ++m)*

*pthread\_join(kids[m].id, NULL);*

*printf("End of Program. Grand Total = %d\n", tot\_items);*

*return 0;*

*}*