In this homework, I did not use any busy-waiting or polling systems, I benefited from semaphores and shared memory only. All signaling operations mentioned in the below pseudocode are made with semaphore operations.

Commands for compiling and running my code:

* compile:

gcc -std=c99 hw3.c -o hw3

* example run:

./hw3 150 4 2 2 4

I have tested my code in ITU SSH server, it works correctly.

Pseudocode for increaser and decreaser is in the next page (let *p* be the current process):

if p is Increaser{

if turn == increasers\_turn:

for increasers\_current\_turn to increasers\_current\_turn + ti:

lock shared memory

increase the money inside the shared memory by 10 or 15

release lock of the shared memory

signal that p has finished its job

if other increasers have also finished:

if (current\_money >= N) and (this is final iteration of for loop):

turn = decreasers\_turn

signal all increasers have finished their job

else:

wait for other increasers to finish

}

if p is Decreaser{

if turn == decreasers\_turn

for decreasers\_current\_turn to decreasers\_current\_turn + td:

lock shared memory

if (p is even decreaser and current\_money is even)

or (p is odd decreaser and current\_money is odd):

if current\_money <= the amount to be subtracted:

signal master process to finish

decrease the money inside the shared memory by fib(p’s\_fib\_index)

release lock of the shared memory

signal that p has finished its job

if other increasers have also finished:

if this is final iteration of for loop:

turn = increasers\_turn

signal all increasers have finished their job

else:

wait for other increasers to finish

}