CIS657 Fall 2018
Assignment Disclosure Form

Assignment #: 3

Name: Nidhi Surya Prakash

1. Did you consult with anyone other than instructor or TA/grader on parts of this assignment?
If Yes, please give the details.

Yes, I did consult.
Malhar Ujawane

Namrata Arkalgud

2. Did you consult an outside source such as an Internet forum or a book on parts of this assignment?
If Yes, please give the details.
References:

1) https://www.student.cs.uwaterloo.ca/~cs350/common/NachosTutorialF06.pdf
2) https://inst.eecs.berkeley.edu/~cs162/fa13/Nachos/nachos.pdf
3) https://en.wikipedia.org/wiki/Job_scheduler

I assert that, to the best of my knowledge, the information on this sheet is true.

Signature:___Nidhi Surya Prakash_____ Date :__5/11/2018

CIS 657 Lab 3

Nidhi Surya Prakash
SUID: 215895619

Objective:

The main objective of this assignment would be to furthermore understand and learn the concepts of C++ in Nachos by adding multithreading capabilities to Nachos whilst making some extensions to the system. We know there are 2 points of view: 1) Unix point of view where Nachos is a process with a user space threads and 2) Nachos point of view where it considers itself as an Operating System which has processes without the multithreading functionality. Here, in this assignment we have 3 tasks – creation of a class named Process, Integration of this class with the Nachos Threads and Nachos System and last, changing the scheduler according to the conveniences.

What has been developed:

Process.h and Process.cc

In these files, we've created a new class named Process. This file is under the thread's directory. The process class contains a constructor and a destructor, also has member variables along with their getters and setters and functions of its own too. We have determined a few member variables and functions.

We have defined our own enum to keep track of the states of the process called processStates. Status variable holds the current status of the process. PID (process ID) which contains the ID of a particular process which goes alongside the pidcounter variable pidC. Priority variable – priority to assign each process its priority based on which it gets scheduled. name – for Name of process and also, funcPtr – function pointer of a function and *args – arguments. A list for all child processes - childProcess

Process.h – used to declare all member variables and functions.

Code:

```
#ifndef NACHOS_PROCESS_H
#define NACHOS_PROCESS_H

#pragma once
#include "thread.h"
#include "../lib/list.h"
#include "scheduler.h"
#include "kernel.h"
#include "../lib/copyright.h"
```

```
#include "../lib/utility.h"
#include "../lib/sysdep.h"
#include "../machine/machine.h"
#include "addrspace.h"


enum processStates { PJUST_CREATED, PRUNNING, PREADY, PBLOCKED };

class Thread;
class Scheduler;
class Kernel;

class Process {
    int priority;
    processStates status; //for states - RUNNING,READY,JUST CREATED,BLOCKED
    char* name;
    int PID;
    static int pidC;
    List<Process*> *childProcess;
    VoidFunctionPtr funcPtr;
    void *args;

public:
    Process(int priority, char* name);
    Process();
    virtual ~Process();

    Scheduler *processScheduler;
    Thread *currentThread;

   //setters declared
    void setStatus(processStates st) { status = st; }
    void setPriority(int p);
    void setPID(int i);
    void setName(char* c);
    //void ProcessPrint(Process *p) { p->Print(); }
```

```
//getters declared
processStates getStatus();
int getPriority();
int getPID();
char* getName() { return (name); }
//int getPriority() const { return priority; };
List<Thread*> *childThreads;
List<Thread*> getchildThreads() { return *childThreads; }
List<Process*> getchildProcess() { return *childProcess; }

//functions used from thread.cc
void Fork(VoidFunctionPtr func, void *arg);
void Terminate();
void Sleep(bool finishing);
void Print() {
    cout << name<<"  "<<endl;
}
void Yield();

//int getPid() const { return PID; };

void createChildT();
void createChildP();
int parentID;
bool isCompleted;

VoidFunctionPtr getfunc()
{
    return funcPtr;
}
void* getArg()
{
    return args;
}

bool operator == (Process p) {
    if(this->getPID() == p.getPID())
```

```
            return true;
        else return false;
    }

    //void Join();
};

extern void ProcessPrint(Process *p);

#endif //NACHOS_PROCESS_H
```

Process.cc – Defines all the functions, getters & setters that are required by the class.

Code:

```
#include "../lib/debug.h"
#include "../machine/interrupt.h"
#include "process.h"
#include "../lib/copyright.h"
#include "thread.h"
#include "switch.h"
#include "synch.h"
#include "../lib/sysdep.h"
#include "processScheduler.h"
#include "../lib/debug.h"
```

```
int Process::pidC=0;

void ProcessPrint(Process *p) { p->Print(); }

//Process constructor
Process::Process(int priority, char* name) {

    cout << "\nIn "<<name<<"'s constructor\n"<<endl;
    this->priority = priority;
    //cout<<"process "<<name<< " has priority "<<getPriority<<endl;
    this->name = name;
    Thread *t = new Thread("main Process thread");
    this->currentThread = t;
    status = PJUST_CREATED;
    PID = pidC++;
    this->processScheduler = new Scheduler();
    childThreads = new List<Thread*>();
    childProcess = new List<Process*>();
    //DEBUG(dbgThread, "Forking Process: " << name << " f(a): " << (int) func << " " << arg);

    cout<<"\n================ Process info
============================================================================================
====================="<<endl;
    cout<<"Process name \t\t\t process PID \t\t\t process priority \t\t\t process state"<<endl;
    cout<<name<<"\t\t\t\t"<<PID<<" \t\t\t\t"<<priority<<"\t\t\t\t\t"<<status<<endl;

cout<<"\n============================================================================================
============================================================="<<endl;
}

/*void setStatus(processStates st)
{
    status = st;
}*/
processStates Process::getStatus(){
    return status;
}
```

```
void Process::setPriority(int p){
    priority = p;
}
int Process::getPriority(){
    return priority;
}
void Process::setPID(int i){
    PID = i;
}
int Process::getPID(){
    return PID;
}
void Process::setName(char* c){
    name = c;
}

//Fork method for process
void Process::Fork(VoidFunctionPtr func, void *arg) {

    cout << "In process "<<this->name<<"'s fork\n";
    Interrupt *interrupt = kernel->interrupt;
    ProcessScheduler *scheduler = kernel->scheduler;

    IntStatus oldLevel;
    currentThread->Fork(this, (VoidFunctionPtr) func, (void *) arg);
    childThreads->Append(currentThread);

    Thread *t1 = new Thread("\nforked thread 1");
    t1->Fork(this,func,arg);
    Thread *t2 = new Thread("\nforked thread 2");
    t2->Fork(this,func,arg);
    Thread *t3 = new Thread("\nforked thread 3");
    t3->Fork(this,func,arg);


    oldLevel = interrupt->SetLevel(IntOff);
    this->currentThread = this->processScheduler->FindNextToRun();
```

```
    //oldLevel = interrupt->SetLevel(IntOff);

    scheduler->ReadyToRun(this);
    scheduler->Print();

    //kernel->currentProcess->Yield();
    //this->pScheduler->ReadyToRun(this->currentThread);

    (void) interrupt->SetLevel(oldLevel);

    funcPtr = func;
    args = arg;
}

//handles creation od child threads
void
Process::createChildT() {
    VoidFunctionPtr func = this->getfunc();
    void* arg = this->getArg();
    Thread *t = new Thread("\nChild Thread\n");
    cout<<"created child thread\n";
    t->Fork(this, func, arg);
    //cout<<"fork of new child thread\n";
    childThreads->Append(t);
}

//handles creation od child processes
void
Process::createChildP() {
    Process *p = new Process(this->priority, "Child process");
    p->parentID = this->PID;
    p->Fork(this->funcPtr, this->args);
    childProcess->Append(p);
    cout<<"\n*******Child process info ******";
    cout<<"\nPriority: "<<priority<<"\n parentID: "<<this->PID<<"\n";
}
```

```
// finish process modified for process
void Process::Terminate() {
    cout << "Process Termination\n";
    (void) kernel->interrupt->SetLevel(IntOff);
    ASSERT(this == kernel->currentProcess);
    DEBUG(dbgThread, "Terminating Process: " << name);
    isCompleted = true;
    Sleep(TRUE);
}

// yeild process defined for process
void
Process::Yield ()
{
    Process *nextProcess;
    IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);

    ASSERT(this == kernel->currentProcess);

    DEBUG(dbgThread, "Yielding process: " << name);
    kernel->scheduler->Print();
    nextProcess = kernel->scheduler->FindNextProcessToRun();
    if (nextProcess != NULL) {

        kernel->scheduler->ReadyToRun(this);
        kernel->scheduler->Run(nextProcess, FALSE);
            //this->currentThread->Yield();
    }
    (void) kernel->interrupt->SetLevel(oldLevel);
}

//sleep function modified for process
void
Process::Sleep (bool finishing)
{
    cout << "process sleep\n";
    Process *nextProcess;
```

```
    ASSERT(this == kernel->currentProcess);
    ASSERT(kernel->interrupt->getLevel() == IntOff);

    status = PBLOCKED;
    while ((nextProcess = kernel->scheduler->FindNextProcessToRun()) == NULL)
        kernel->interrupt->Idle();  // no one to run, wait for an interrupt

    // returns when it's time for us to run
    kernel->scheduler->Run(nextProcess, finishing);
    //this->currentThread->Sleep(finishing);
}

//Process destructor
Process::~Process() {
    delete childThreads;
    delete childProcess;
    delete processScheduler;
    delete currentThread;
}
```

Changes made and additions:

thread.h

```
class Scheduler;
class Process;
```

```
void Fork(Process *p, VoidFunctionPtr func, void *arg);
          // Make thread run (*func)(arg)
    void Yield();     // Relinquish the CPU if any
        // other thread is runnable
    void Sleep(bool finishing); // Put the thread to sleep and
```

```
    // relinquish the processor
  void Begin();   // Startup code for the thread
  void Finish();      // The thread is done executing

  void CheckOverflow();     // Check if thread stack has overflowed
  void setStatus(ThreadStatus st) { status = st; }
  char* getName() { return (name); }
  void Print() { cout << name<<" "; }
  void SelfTest();    // test whether thread impl is working
  bool isCompleted;
```

thread.cc – here functions have been modified so that we can implement multithreading and process.

Changes and additions made:

```
Thread::~Thread()
{
  DEBUG(dbgThread, "Deleting thread: " << name);

  ASSERT(this != kernel->currentProcess->currentThread);
  if (stack != NULL)
  DeallocBoundedArray((char *) stack, StackSize * sizeof(int));
}
```

```
Thread::Fork(Process *p, VoidFunctionPtr func, void *arg)
{

  Interrupt *interrupt = kernel->interrupt;
  // Scheduler *scheduler = kernel->scheduler;
  IntStatus oldLevel;

  DEBUG(dbgThread, "Forking thread: " << name << " f(a): " << (int) func << " " << arg);

  StackAllocate(func, arg);

  oldLevel = interrupt->SetLevel(IntOff);
  p->processScheduler->ReadyToRun(this);  // ReadyToRun assumes that interrupts
```

```
    (void) interrupt->SetLevel(oldLevel);
}
```

```
void
Thread::Begin ()
{
    ASSERT(this == kernel->currentProcess->currentThread);
    DEBUG(dbgThread, "Beginning thread: " << name);
    cout << "Beginning thread: " << name << endl;

    kernel->scheduler->CheckToBeDestroyed();
    kernel->interrupt->Enable();
}
```

```
void
Thread::Finish ()
{
    (void) kernel->interrupt->SetLevel(IntOff);
    ASSERT(this == kernel->currentProcess->currentThread);

    DEBUG(dbgThread, "Finishing thread: " << name);
    cout << "In thread finish. Finishing thread: " << name << endl;
    isCompleted = true;

    Sleep(TRUE);                // invokes SWITCH
    // not reached
}
```

```
void
Thread::Yield ()
{
    Thread *nextThread;
    IntStatus oldLevel = kernel->interrupt->SetLevel(IntOff);
```

```
    ASSERT(this == kernel->currentProcess->currentThread);

    DEBUG(dbgThread, "Yielding thread: " << name);
    cout << "Yielding thread: " << name << endl;

    nextThread = kernel->currentProcess->processScheduler->FindNextToRun();
    if (nextThread != NULL) {
    kernel->currentProcess->processScheduler->ReadyToRun(this);
    kernel->currentProcess->processScheduler->Run(nextThread, FALSE);
    } else {
        bool allDone = true;

        ListIterator<Thread*> *li = new ListIterator<Thread*>(kernel->currentProcess->childThreads);
        while (!li->IsDone()) {
            if(!li->Item()->isCompleted)
                allDone = false;
            li->Next();
        }

        if(allDone)
            kernel->currentProcess->Terminate();
    }
    (void) kernel->interrupt->SetLevel(oldLevel);
}
```

```
void
Thread::Sleep (bool finishing)
{
    Thread *nextThread;

    ASSERT(this == kernel->currentProcess->currentThread);
    ASSERT(kernel->interrupt->getLevel() == IntOff);

    DEBUG(dbgThread, "Sleeping thread: " << name);
    cout << "Sleeping thread: " << name << endl;
```

```
    status = BLOCKED;
    while ((nextThread = kernel->currentProcess->processScheduler->FindNextToRun()) == NULL)
        //kernel->interrupt->Idle();    // no one to run, wait for an interrupt
        kernel->currentProcess->Terminate();
    // returns when it's time for us to run
    kernel->currentProcess->processScheduler->Run(nextThread, finishing);
}
```

```
static void ThreadFinish()    { kernel->currentProcess->currentThread->Finish(); }
static void ThreadBegin() { kernel->currentProcess->currentThread->Begin(); }
void ThreadPrint(Thread *t) { t->Print(); }
```

threadTest.cc –

Changes and additions made:

```
#include "kernel.h"
#include "main.h"
#include "process.h"

void
SimpleThread(int which)
{
    int num;

    for (num = 0; num < 5; num++) {
        printf("*** thread %d looped %d times\n", which, num);
    }
    //kernel->currentProcess->currentThread->Yield();
    kernel->currentProcess->Terminate();

   // }
}

void
SimplePrint(int loop){
```

```
    for(int i=0;i<2; i++)
    cout << "Hello this is process : " << loop << endl;
}

void
ThreadTest()
{
    cout << "\n\nIn threadtest main\n";
    Process *p1 = new Process(4, "Process p1");
    p1->Fork((VoidFunctionPtr) SimpleThread, (void *) 1);

    Process *p2 = new Process(3, "Process p2");
    p2->Fork((VoidFunctionPtr) SimpleThread, (void *) 1);

    Process *p3 = new Process(1, "Process p3");
    p3->Fork((VoidFunctionPtr) SimpleThread, (void *) 1);

    Process *p4 = new Process(5, "Process p4");
    p4->Fork((VoidFunctionPtr) SimpleThread, (void *) 1);

    Process *p5 = new Process(2, "Process p5");
    p5->Fork((VoidFunctionPtr) SimpleThread, (void *) 1);


    p1->createChildP();
    p1->createChildT();


    p2->createChildP();
    p2->createChildT();

    p3->createChildP();
    p3->createChildT();

    p4->createChildP();
    p4->createChildT();
```

```
    p5->createChildP();
    p5->createChildT();

    kernel->currentProcess->Terminate();
}
```

processScheduler.cc and processScheduler.h – These files include a few new functions that have been declared and defined and few functions have been taken from the thread scheduler – scheduler.cc and scheduler.h with modifications done accordingly so as to help in implementing the scheduler explicitly for the process class. Here we have also used a sortedList to processes.

Changes and additions made:

processScheduler.h

```
#ifndef PROCESSSCHEDULER_H
#define PROCESSSCHEDULER_H

#pragma once
#include "../lib/copyright.h"
#include "../lib/list.h"
#include "process.h"

class Process;
class ProcessScheduler {
  public:
    ProcessScheduler();
    ~ProcessScheduler();
    void ReadyToRun(Process* process);
    Process* FindNextProcessToRun();
    void Run(Process* nextProcess, bool finishing);
    void CheckToBeDestroyed();
    void Print();
    void CheckIfParentUnblocked();
  private:

    SortedList<Process *> *pReadyList;
```

```
    Process *pToBeDestroyed;
};

#endif
```

processScheduler.cc

```
int comp(Process *p1, Process *p2){

    if(p1->getPriority() < p2->getPriority())
        return 1;
    else if(p1->getPriority() > p2->getPriority())
        return -1;
    else return 0;
}

ProcessScheduler::ProcessScheduler()
{
    pReadyList = new SortedList<Process *>(comp);
    pToBeDestroyed = NULL;
}
```

```
ProcessScheduler::~ProcessScheduler()
{
    delete pReadyList;
}
```

```
void
ProcessScheduler::ReadyToRun (Process *process)
{
    cout<<"\nIn "<<process->getName()<<" ready to run \n";
    ASSERT(kernel->interrupt->getLevel() == IntOff);
    DEBUG(dbgThread, "Putting process on ready list: " << process->getName());
    cout << "Pushing process on readylist: " << process->getName() << endl;
    process->setStatus(PREADY);
```

```
    pReadyList->Insert(process);
}
```

```
Process *
ProcessScheduler::FindNextProcessToRun ()
{
    //cout << "In process find next to run\n";
    ASSERT(kernel->interrupt->getLevel() == IntOff);

    if (pReadyList->IsEmpty()) {
    return NULL;
    }
    else {
        return pReadyList->RemoveFront();
    }
}
```

```
void
ProcessScheduler::Run (Process *nextProcess, bool finishing)
{
    Process *oldProcess = kernel->currentProcess;
    cout << "In process "<<oldProcess->getName()<<" run with priority"<<oldProcess->getPriority()<<"\n";
    Thread * oldthead = kernel->currentProcess->currentThread;

    ASSERT(kernel->interrupt->getLevel() == IntOff);

    if (finishing) {    // mark that we need to delete current thread
        //ASSERT(processToBeDestroyed == NULL);
     //processToBeDestroyed = oldProcess;
    }

    kernel->currentProcess = nextProcess;  // switch to the next thread
    nextProcess->setStatus(PRUNNING);      // nextThread is now running

    Thread *nextThread = nextProcess->currentThread;
    DEBUG(dbgThread, "Switching from: " << oldProcess->getName() << " to: " << nextProcess->getName());
```

```
    kernel->currentProcess->currentThread = nextThread;
    nextProcess->currentThread->setStatus(RUNNING);
    SWITCH(oldthead, nextThread);

    ASSERT(kernel->interrupt->getLevel() == IntOff);

    DEBUG(dbgThread, "Now in thread: " << oldProcess->getName());

    CheckToBeDestroyed();   // check if thread we were running
}
```

```
void
ProcessScheduler::CheckToBeDestroyed()
{
    if (pToBeDestroyed != NULL) {
        delete pToBeDestroyed;
    pToBeDestroyed = NULL;
    }
}

void
ProcessScheduler::Print()
{
    cout << "\nProcess readylist has \n";
    pReadyList->Apply(ProcessPrint);
}
```

Kernel.h and kernel.cc – In kernel.h we made a few changes and in kernel.cc we have created a new process main and also contains the code for the quantum flag implementation.

Changes and additions made:

Kernel.h

```
Process *currentProcess;
    ProcessScheduler *scheduler;
```

```
List<Process*> *proclist = new List<Process*>();
    int quantum;
```

Kernel.cc

```
cout << "***********Kernel has been initialized ******************\n";
    cout<<"\nBegins at kernel";
    currentProcess = new Process(0, "main");
    currentProcess->setStatus(PRUNNING);
```

Scheduler.h and scheduler.cc – scheduler for threads.

Changes and additions made:

Scheduler.cc

```
void
Scheduler::Run (Thread *nextThread, bool finishing)
{
 Thread *oldThread = kernel->currentProcess->currentThread;
 cout << "In thread "<<oldThread->getName()<< "run\n";
 ASSERT(kernel->interrupt->getLevel() == IntOff);

 if (finishing) {   // mark that we need to delete current thread
 ASSERT(toBeDestroyed == NULL);
toBeDestroyed = oldThread;
}

 if (oldThread->space != NULL) {    // if this thread is a user program,
  oldThread->SaveUserState();   // save the user's CPU registers
```

```
    oldThread->space->SaveState();
  }

oldThread->CheckOverflow();         // check if the old thread had an undetected stack overflow

 kernel->currentProcess->currentThread = nextThread;  // switch to the next thread
 nextThread->setStatus(RUNNING);      // nextThread is now running

DEBUG(dbgThread, "Switching from: " << oldThread->getName() << " to: " << nextThread->getName());

SWITCH(oldThread, nextThread);

ASSERT(kernel->interrupt->getLevel() == IntOff);

DEBUG(dbgThread, "Now in thread: " << oldThread->getName());

CheckToBeDestroyed

if (oldThread->space != NULL) {     // if there is an address space
  oldThread->RestoreUserState();     // to restore, do it.
oldThread->space->RestoreState();
 }
}
```

Scheduler.h

```
class Thread;
```

Synch.h and synch.cc

Changes and additions:

Synch.h

```
bool IsHeldByCurrentThread() {
       return lockHolder == kernel->currentProcess->currentThread; }
```

```
        // return true if the current thread
```

synch.cc

```
void
Semaphore::P()
{
    Interrupt *interrupt = kernel->interrupt;
    Thread *currentThread = kernel->currentProcess->currentThread;

    // disable interrupts
    IntStatus oldLevel = interrupt->SetLevel(IntOff);

    while (value == 0) {        // semaphore not available
    queue->Append(currentThread);   // so go to sleep
    currentThread->Sleep(FALSE);
    }
    value--;            // semaphore available, consume its value

    // re-enable interrupts
    (void) interrupt->SetLevel(oldLevel);
}
```

```
void
Semaphore::V()
{
    Interrupt *interrupt = kernel->interrupt;

    // disable interrupts
    IntStatus oldLevel = interrupt->SetLevel(IntOff);

    if (!queue->IsEmpty()) {  // make thread ready.
    kernel->currentProcess->processScheduler->ReadyToRun(queue->RemoveFront());
    }
    value++;
```

```
    // re-enable interrupts
    (void) interrupt->SetLevel(oldLevel);
}
```

```
void Lock::Acquire()
{
    semaphore->P();
    lockHolder = kernel->currentProcess->currentThread;
}
```

Timer.h and timer.cc – changes are made to handle the quantum

Changes and additions:

timer.cc

```
quant = quantum;
```

```
void
Timer::SetInterrupt()
{
    if (!disable) {
        int delay = this->quant;

        if (randomize) {
          delay = 1 + (RandomNumber() % (this->quant * 2));
         }
        // schedule the next timer device interrupt
        kernel->interrupt->Schedule(this, delay, TimerInt);
    }
}
```

Timer.h

```
int quant;
```

How to test your solution

Step 1: Login with your server link and password. Start filezilla using the same credentials.
Step 2: Go into the 'code' folder of the 'nachos' folder. Access 'build.linux' folder.
Step 3: Run the following commands:

```
make distclean
make depend
make nachos
```

Step 4: Make sure the updated files are uploaded using filezilla
Step 5: Run the command: ./nachos -K

Files modified / Added: Give Fully qualified path of the files Added:

Modified:

1) threadtest.cc
2) main.cc
3) main.h
4) kernel.h
5) kernel.cc
6) scheduler.h
7) scheduler.cc
8) synch.h
9) synch.cc
10) alarm.cc
11) alarm.h
12) timer.cc
13) thread.cc
14) thread.h
15) addrspace.cc
16) stats.cc
17) stats.h
18) Makefile
19) interrup.cc
20) mipssim.cc

Added:

1) Process.cc
   nachos/code/threads/process.cc
2) Process.h
   nachos/code/threads/process.h
3) processScheduler.cc
   nachos/code/threads/processScheduler.cc
4) processScheduler.h

nachos/code/threads/processScheduler.h

Outputs:

```
[nsuryapr@lcs-vc-cis486:~/PA1_new/nachos_PA1_final/code/build.linux$ ./nachos -K ]
************Kernel has been initialized ******************

In main's constructor


============== Process info ============================================================================================================
Process name                    process PID                    process priority                    process state
main                           0                             0                                   0

========================================================================================================================================

In postal worker's constructor


============== Process info ============================================================================================================
Process name                    process PID                    process priority                    process state
postal worker                          1                              1                                   0

========================================================================================================================================
In process postal worker's fork

Putting thread on ready list
 Thread readylist now has:
 main Process thread
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
forked thread 3
In postal worker ready to run
Pushing process on readylist: postal worker

Process readylist has
postal worker


In threadtest main

In Process p1's constructor


============== Process info ============================================================================================================
Process name                    process PID                    process priority                    process state
Process p1                             2                              4                                   0
```

```
================================================================================================================================
In process Process p1's fork

Putting thread on ready list
 Thread readylist now has:
 main Process thread
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
forked thread 3
In Process p1 ready to run
Pushing process on readylist: Process p1

Process readylist has
Process p1
postal worker

In Process p2's constructor


============== Process info ====================================================================================================
Process name                    process PID                    process priority                    process state
Process p2                              3                              3                                   0

================================================================================================================================
In process Process p2's fork

Putting thread on ready list
 Thread readylist now has:
 main Process thread
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
```

```
Putting thread on ready list
 Thread readylist now has:
 main Process thread
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
forked thread 3
In Process p2 ready to run
Pushing process on readylist: Process p2

Process readylist has
Process p1
Process p2
postal worker

In Process p3's constructor


============== Process info ====================================================================================================
Process name                    process PID                    process priority                    process state
Process p3                              4                              1                                   0

================================================================================================================================
In process Process p3's fork

Putting thread on ready list
 Thread readylist now has:
 main Process thread
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
forked thread 3
In Process p3 ready to run
Pushing process on readylist: Process p3
```

```
In Process p3 ready to run
Pushing process on readylist: Process p3

Process readylist has
Process p1
Process p2
postal worker
Process p3

In Process p4's constructor


=============== Process info ================================================================================================================
Process name                    process PID                    process priority                    process state
Process p4                             5                              5                                   0

============================================================================================================================================
In process Process p4's fork

Putting thread on ready list
 Thread readylist now has:
 main Process thread
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
forked thread 3
In Process p4 ready to run
Pushing process on readylist: Process p4

Process readylist has
Process p4
Process p1
Process p2
postal worker
Process p3

In Process p5's constructor


=============== Process info ================================================================================================================
Process name                    process PID                    process priority                    process state
Process p5                             6                              2                                   0

============================================================================================================================================
In process Process p5's fork
```

```
============== Process info ============================================================================================================
Process name                    process PID                    process priority                    process state
Process p5                             6                              2                                   0

========================================================================================================================================
In process Process p5's fork

Putting thread on ready list
 Thread readylist now has:
 main Process thread
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
forked thread 3
In Process p5 ready to run
Pushing process on readylist: Process p5

Process readylist has
Process p4
Process p1
Process p2
Process p5
postal worker
Process p3

In Child process's constructor


============== Process info ============================================================================================================
Process name                    process PID                    process priority                    process state
Child process                          7                              4                                   0

========================================================================================================================================
In process Child process's fork

Putting thread on ready list
 Thread readylist now has:
 main Process thread
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
Putting thread on ready list
 Thread readylist now has:
```

```
Putting thread on ready list
 Thread readylist now has:
 main Process thread
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
forked thread 3
In Child process ready to run
Pushing process on readylist: Child process

Process readylist has
Process p4
Process p1
Child process
Process p2
Process p5
postal worker
Process p3

*******Child process info ******
Priority: 4
 parentID: 2
created child thread

Putting thread on ready list
 Thread readylist now has:

forked thread 1
forked thread 2
forked thread 3
Child Thread

In Child process's constructor


============== Process info ============================================================================================================
Process name                    process PID                    process priority                    process state
Child process                          8                              3                                   0

========================================================================================================================================
In process Child process's fork

Putting thread on ready list
 Thread readylist now has:
```

```
In Child process's constructor


============== Process info ================================================================================================================
Process name                    process PID                    process priority                    process state
Child process                          8                              3                                   0

============================================================================================================================================
In process Child process's fork

Putting thread on ready list
 Thread readylist now has:
 main Process thread
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
forked thread 3
In Child process ready to run
Pushing process on readylist: Child process

Process readylist has
Process p4
Process p1
Child process
Process p2
Child process
Process p5
postal worker
Process p3

*******Child process info ******
Priority: 3
 parentID: 3
created child thread

Putting thread on ready list
 Thread readylist now has:

forked thread 1
forked thread 2
forked thread 3
Child Thread

In Child process's constructor
```

```
In Child process's constructor


============== Process info ================================================================================================================
Process name                    process PID                    process priority                    process state
Child process                          9                              1                                   0

============================================================================================================================================
In process Child process's fork

Putting thread on ready list
 Thread readylist now has:
 main Process thread
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
forked thread 3
In Child process ready to run
Pushing process on readylist: Child process

Process readylist has
Process p4
Process p1
Child process
Process p2
Child process
Process p5
postal worker
Process p3
Child process

*******Child process info ******
Priority: 1
 parentID: 4
created child thread

Putting thread on ready list
 Thread readylist now has:

forked thread 1
forked thread 2
forked thread 3
Child Thread
```

```
forked thread 2
forked thread 3
Child Thread

In Child process's constructor


============== Process info ================================================================================================================
Process name                    process PID                    process priority                    process state
Child process                          10                             5                                   0

============================================================================================================================================
In process Child process's fork

Putting thread on ready list
 Thread readylist now has:
 main Process thread
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
forked thread 3
In Child process ready to run
Pushing process on readylist: Child process

Process readylist has
Process p4
Child process
Process p1
Child process
Process p2
Child process
Process p5
postal worker
Process p3
Child process

*******Child process info ******
Priority: 5
 parentID: 5
created child thread

Putting thread on ready list
 Thread readylist now has:

forked thread 1
```

```
*******Child process info ******
Priority: 5
 parentID: 5
created child thread

Putting thread on ready list
 Thread readylist now has:

forked thread 1
forked thread 2
forked thread 3
Child Thread

In Child process's constructor


============== Process info ================================================================================================================
Process name                    process PID                    process priority                    process state
Child process                          11                             2                                   0

============================================================================================================================================
In process Child process's fork

Putting thread on ready list
 Thread readylist now has:
 main Process thread
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
Putting thread on ready list
 Thread readylist now has:
 main Process thread
forked thread 1
forked thread 2
forked thread 3
In Child process ready to run
Pushing process on readylist: Child process

Process readylist has
Process p4
Child process
Process p1
Child process
Process p2
Child process
Process p5
Child process
postal worker
Process p3
Child process
```

```
*******Child process info ******
Priority: 2
 parentID: 6
created child thread

Putting thread on ready list
 Thread readylist now has:

forked thread 1
forked thread 2
forked thread 3
Child Thread
 Process Termination
process sleep
In process main run with priority0
Beginning thread: main Process thread
*** thread 1 looped 0 times
*** thread 1 looped 1 times
*** thread 1 looped 2 times
*** thread 1 looped 3 times
*** thread 1 looped 4 times
Process Termination
process sleep
In process Process p4 run with priority5
Beginning thread: main Process thread
*** thread 1 looped 0 times
*** thread 1 looped 1 times
*** thread 1 looped 2 times
*** thread 1 looped 3 times
*** thread 1 looped 4 times
Process Termination
process sleep
In process Child process run with priority5
Beginning thread: main Process thread
*** thread 1 looped 0 times
*** thread 1 looped 1 times
*** thread 1 looped 2 times
*** thread 1 looped 3 times
*** thread 1 looped 4 times
Process Termination
process sleep
In process Process p1 run with priority4
Beginning thread: main Process thread
*** thread 1 looped 0 times
*** thread 1 looped 1 times
*** thread 1 looped 2 times
*** thread 1 looped 3 times
*** thread 1 looped 4 times
Process Termination
process sleep
In process Child process run with priority4
Beginning thread: main Process thread
*** thread 1 looped 0 times
*** thread 1 looped 1 times
*** thread 1 looped 2 times
```

```
process sleep
In process Child process run with priority4
Beginning thread: main Process thread
*** thread 1 looped 0 times
*** thread 1 looped 1 times
*** thread 1 looped 2 times
*** thread 1 looped 3 times
*** thread 1 looped 4 times
Process Termination
process sleep
In process Process p2 run with priority3
Beginning thread: main Process thread
*** thread 1 looped 0 times
*** thread 1 looped 1 times
*** thread 1 looped 2 times
*** thread 1 looped 3 times
*** thread 1 looped 4 times
Process Termination
process sleep
In process Child process run with priority3
Beginning thread: main Process thread
*** thread 1 looped 0 times
*** thread 1 looped 1 times
*** thread 1 looped 2 times
*** thread 1 looped 3 times
*** thread 1 looped 4 times
Process Termination
process sleep
In process Process p5 run with priority2
Beginning thread: main Process thread
*** thread 1 looped 0 times
*** thread 1 looped 1 times
*** thread 1 looped 2 times
*** thread 1 looped 3 times
*** thread 1 looped 4 times
Process Termination
process sleep
In process Child process run with priority2
Beginning thread: main Process thread
Sleeping thread: main Process thread
In thread main Process threadrun
Beginning thread:
forked thread 1
Sleeping thread:
forked thread 1
In thread
forked thread 1run
Beginning thread:
forked thread 2
Sleeping thread:
forked thread 2
In thread
forked thread 2run
Beginning thread:
forked thread 3
Sleeping thread:
```

```
*** thread 1 looped 1 times
*** thread 1 looped 2 times
*** thread 1 looped 3 times
*** thread 1 looped 4 times
Process Termination
process sleep
In process Process p5 run with priority2
Beginning thread: main Process thread
*** thread 1 looped 0 times
*** thread 1 looped 1 times
*** thread 1 looped 2 times
*** thread 1 looped 3 times
*** thread 1 looped 4 times
Process Termination
process sleep
In process Child process run with priority2
Beginning thread: main Process thread
Sleeping thread: main Process thread
In thread main Process threadrun
Beginning thread:
forked thread 1
Sleeping thread:
forked thread 1
In thread
forked thread 1run
Beginning thread:
forked thread 2
Sleeping thread:
forked thread 2
In thread
forked thread 2run
Beginning thread:
forked thread 3
Sleeping thread:
forked thread 3
Process Termination
process sleep
In process postal worker run with priority1
Beginning thread: main Process thread
*** thread 1 looped 0 times
*** thread 1 looped 1 times
*** thread 1 looped 2 times
*** thread 1 looped 3 times
*** thread 1 looped 4 times
Process Termination
process sleep
In process Process p3 run with priority1
Beginning thread: main Process thread
*** thread 1 looped 0 times
*** thread 1 looped 1 times
*** thread 1 looped 2 times
*** thread 1 looped 3 times
*** thread 1 looped 4 times
Process Termination
process sleep
```