OS-PG PINTOS ASSIGNMENT

Submitted By:Dhawal Jain(0801IT141028)

PART 1

1.Insallation the given pintos

=> Step one: **Install Qemu** on your machine .

Type the following command in your terminal to install Qemu.

sudo apt-get install qemu

=> Step two: **Download Pintos** on your machine.

=> Step three: **Extract** the pintos.tar.gz file.

Create some directory in your $HOME , say os-pintos (this is followed throughout the article) using the following command.

mkdir os-pintos

Change to this directory from home using the following command.

cd os-pintos

First copy the pintos.tar.gz file from the downloads into this directory.Extract the “pintos.tar.gz” file in this folder using the following command.

tar -xvzf pintos.tar.gz

=> Step four: Set **GDBMACROS**

Open the file “pintos-gdb” which is present in /$HOME/os-pintos/pintos/src/utils . Find the variable GDBMACROS and set it to make it point to path‘$HOME/os-pintos/pintos/src/misc/gdb-macros’. The commands used are given below.

cd /pintos/src/utils

gedit pintos-gdb

change path of “*GDBMACROS=$HOME/os-pintos/pintos/src/misc/gdb-macros*“.

=> Step five: **Edit Makefile** in utils directory

Open the “Makefile” in the utils folder and replace line number five , “LDFLAGS = -lm” by “LDLIBS = -lm”.

=> Step six: **Compile the utilities** in the “utils” folder .

Use the following command to compile the utilities folder (assuming you are still in utils folder).

make

=> Step seven: **Edit Make.vars** in threads directory

Open the file Make.vars in “/home/dhawal/*os-pintos/pintos/src/threads/Make.vars*” and change the last line(line number seven) to:

“SIMULATOR = –qemu”

PART 2

PREEMTION OF THREAD

FILES AND FUNCTION MODIFIED

File 1: thread.h

Update struct thread to include new vairable that stored number of ticks to sleep i.e ***time_to_sleep*** and a list-elem ***alarm_for_wake*** which will be used as a thread member in blocked state.

FILE 2:timer.c

Method:timer_sleep()

Recieving time a thread need to sleep if time i.e ticks > 0 than
calling thread_sleep in thread.c

File 3:thread.c

Method:thread_init()

Initialize blocked_state list in thread_init and in order to ensure that thread wake will be called after whole system iniatialize a variable named initialize is made true and initial thread sleep time=0.

Method: thread_sleep()

Defining the sleep time of the current thread and pushing in blocked_state list and blocking the thread

Method: thread_wakeup()

Traverse the sleep list i.e blocked_state and check if thread
has slept enough than remove it from the blocked state

Test cases passed:

alarm-single
alarm-zero
alarm-negative
alarm-multiple
alarm-simultaneous

Logic of Implementation:

Our solution is to create a new variable in thread structure that will store the number of ticks to sleep.
Timer_interrupt is the API that CPU call at each tick.In this function we can iterate over all the threads/processes and check the processes whose ticks to sleep have expired, and wake up that process i.e. thread_unblock() that specific thread.
This prevent busy-waiting and also savescpu from lot of context-switches improving efficieny.

PART 3

IMPLEMENT PRIORITY SCHEDULLING

FILES AND FUNCTION MODIFIED:

File 1: thread.c

Mehod:bool my_priority_picker(const struct list_elem *a, const struct list_elem *b, void *aux)
{struct thread *ta = list_entry(a, struct thread, elem) ;

It is just a compare function for compairing priority of threads

Method:thread_create (const char *name, int priority,thread_func *function, void *aux)

Scheduling if new thread priority is more than current running thread priority

Method:void thread_set_priority (int new_priority)
Set the current thread priority to new_priority and yield the cpu if higher priority process in queue.

Method:static struct thread *next_thread_to_run (void)
Called from schedule to get next highest priority thread from ready queue.

Test Case Passed:

alarm-priority
priority-fifo
priority-change

Implementation Logic:

1.We reuse the list_max api of list.h to find the highest priority thread in ready list.

2.The next_thread_to_run is modified to return highest priority thread

3.We also check the priority of newly created thread and yield the cpu if priority of running thread is lower than the priority of new thread.

4.We also yield the cpu if running thread lowers its priority and there exist a thread in ready list with priority higher than the priority of running thread.

File 2:sync.c

Method:void sema_up (struct semaphore *sema)

Waking up the thread of highest priority from the sema->waiters list instead of just pop from front while sema up.

Test Case Passed:

priority-sema

Implemtation Logic:
Just ensure that highest priority thread only unblock while signal operation of semaphore i.e through semaUp

Method :void cond_signal(struct condition *cond, struct lock *lock UNUSED)

Sorting the condition waiting list and passing correspondingly the front into semaUp.

Method:bool sema_priority_less_helper(const struct list_elem *a,const struct list_elem *b, void *aux);

Comparator function for semaphore representing thread priority which is waiting for a condition to be true.

Test Case Passed:

priority-condvar

Implemtation Logic

Ensuring Priority wise pop from condition waiting list.