ALARM CLOCK
===========

Objective:

Removing busy waiting to implement alarm clock functionality.

In thread.h and thread.c,

1) Added variable to thread struct:
Variable: int64_t wake_up;
Description: wake_up is the time on which the thread finishes it's sleep

2) Added function:
a) void block_threads(struct list * b_list , int64_t wake_up);
Name : block_threads
Input_param : struct list * , int64_t
Output : void
Description : changes the status of the thread and puts it in the block list.

b) void insert_in_list (const struct list * a, struct list_elem * b);
Name : insert_in_list
Input_param : struct list * , struct list_elem *
Output : void
Description : Adds the thread in accending order in the block list depending on their wake_up time.

c) void unblock_threads(struct list * b_list , int64_t now);
Name : unblock_threads
Input_param : struct list * , int64_t
Output : void
Description : Loops through the block_list and checks which threads are ready to run again.

In timer.h and timer.c

1)Added global variable
Variable:struct list block_list;
Description:data structure to maintain list sleeping threads.

2) Changed functions:
timer_sleep
timer_interrupt

Algorithm

In timer_sleep:
1) Calculate the wake_up value.It indicates when the thread is done sleeping and is ready to be unblocked. This is calculated by adding the global ticks (ticks since the OS booted) to the ticks argument.
2) It calls block_threads() which adds the current thread to the sleep list.It is added in sorted order such that the front thread element has the lowest sleeping time.The function implemented for the same is insert_in_list().
3) Changes the status of thread to BLOCKED

In timer interrupt handler:
1) Gets the global ticks from OS to compare.
2) Calls unblock_threads().
3) Checks the first element of the block list, if the wake_up value of element (thread) less than or equal to the global ticks then the thread is removed from the sleep list and is unblocked (Status changed to RUNNING and added to ready_list).
4) Repeat steps 3 until the block list is empty or the thread's wake up value is greater than the global ticks.

PRIORITY SCHEDULING
===================

Ojective:

To implement a scheduling algorithm based on priorities of the threads spawned.

In thread.h and thread.c,

1) Added function's prototype:
a) bool priority_compare(const struct list_elem *a, const struct list_elem *b, void *aux);
Name : priority_compare
Input_param : struct list_elem *,const struct list_elem *,void *
Output : bool
Description : Compare the current thread prirority and other threads priority present in the ready list

b) bool sem_priority_compare(const struct list_elem *a, const struct list_elem *b, void *aux)
Name : sem_priority_compare
Input_param : struct list_elem *,const struct list_elem *,void *
Output : bool
Description : Check for higher priority with respect to condition list declared in synch.c

2) Changed functions:
thread_create(): Check if the new thread has higher priority than currently runnig thread, if so call thread yield()
thread_yield(): Added ordered insertion
thread_unblock(): Added ordered insertion
thread_set_priority(): Added thread yield

In synch.h and synch.h

1)Made local structure semaphore_elem as global structure

2)Changed functions:
sema_down(): Added ordered insertion
sema_up(): Check if the newly unblocked thread has higher priority than currently runnig thread, if so call thread yield()
cond_wait(): Added ordered insertion

Algorithm
----

1) All the threads created have a priority assigned to them.Higher the value highr the priority.
2) The threads are inserted into ready list (list that maintains the order of allocation of CPU to thread) according

to the decreasing order of their priority.
3) Whenever a new thread with higher priority than the currently running thread is spawned or unblocked, the current thread is preempted to allow latter to be scheduled.

Drawback:

The lower priority threads can be starved if the user does not change the priority.

ADVANCED SCHEDULER
==================

Objective:

To implement multi level feedback queue so as to reduce the average response time for running jobs.

Note:To shutdown the simmulator after every test case, small modification has been done in shutdown.c

INSIGHTS/UNDERSTANDING/EXPERIENCE
=================================

1) Closer observation of how OS works for implementing preemptive scheduling.
2) There has been queer observation that, unblocking a thread and later on removing its element from block list results in page faults.
3) Yield process(preemption) is interrupt driven and therefore cannot work if interrupt is disabled.