Design Document Pintos Project(CSL333)

Prateek Garg(P2008CS1024)

Chhavi Mittal(P2008CS1008)

PROJECT 1(THREADS)

Changes for implementing 1.1 Alarm clock-

In Timer.c

- ***compare_block_list (const struct list_elem *a , const struct list_elem *b , void *aux_ UNUSED)***
  it is a simple function to compare two list elements . It is used in many other functions for comparing two elements so as to insert the element in the sorted order.

- **timer_sleep (int64_t ticks)**
  if the wake up time of thread is greater than the current timer ticks , then the thread is put to sleep and inserted in list of blocked threads in order of decreasing wake up time and if its curent status is running, it is set to be blocked. This is implementation is done by disabling interrupts and after finishing, setting the interrupt to its old value.

- **timer_interrupt (struct intr_frame *args UNUSED)**
  it keeps on checking whether the wake up time has become equal to the ticks, if so, thread is waked up and it is removed from blocked list and added to ready queue.

Changes for implementing Synchronization:-

priority donation and scheduling:-

In thread.h
New variables in the structure of the thread

- **int orig_priority;**   /* variable to store the original priority so that after chain donation we can revert back to the original prioritiy*/
- **struct list_elem allelem;**   /* List element for all threads list. */
- **struct lock *thread_lock;**   /* variable of lock type to store the lock on which a thread is blocked*/

void yield_on_priority(void);
/* this functions checks if the priority of the current running thread is less than the priority of the first thread in the ready list. If this is true then thread_yield is called so as to schedule the new thread according to the priority.*/

In thread.c

Changed/modified/improved functions

- **1)thread_init**
  block list is also initialized.

- **2)compare_ready_list** (const struct list_elem *a , const struct list_elem *b , void *aux_ UNUSED)
  This function is used to compare the elements of ready list according to their priority.
- **3) compare_priority**
  This function compares the priority of 2 threads.
- **thread_unblock** (struct thread *t)
  The thread is put in the ready list in ordered manner of priority.
- **thread_set_priority**
  it sets the priority of current current to new prority . It checks if the lock acuired by current thread is zero, then priority is chanegd and yield_on_priority is called.

Apart from these changes many small changed have ben made to certain functions, like thread_get_name, get_priority etc.

In Synch.h

1)struct list_elem holder_elem;
element to be stored in the locks_acquired list of the holder thread. This is declared in struct lock

In Synch.c

- **compare_waiters_list**
  this is used to comapre elements of waiters list.
- **sema_down (struct semaphore *sema)**
  if the value of semaphore is zero then, it is inserted into list of waiting threads and the thread is blocked.
- **sema_up (struct semaphore *sema)**
  Iin this fucntion the list of semaphore waiters is checked and the the waiters are arranged in sorted order.
- **lock_acquire (struct lock *lock)**
  First interrupts are disable, and then if lock holder is null, then lock of current thread is set to 'lock '. Further check is done to see if priority of lock holder is less then priority of current thread, then priority of lock holder is set to priority of current thread.

If status of lock holder is blocked, and further if **lock holder -> thread_lock** is not **Null** then the list of semaphore **waiters** is sorted (of **lock holder -> thread-lock**).
if status of lock holder is ready then, ready list is sorted.
Next in this function, priority chain donation is implemented in the while loop.
Then interrupt is set to old one, and yield_on priority is called.

- **int priority_on_releasing_lock(struct lock *lock)**
  this is the function made by us to assign the priority to the thread when it releass lock its original priority i.e, during priority donation , the priorities were changed, but when thread releases lock, its priority is made equal to original priority.
- **lock_release (struct lock *lock)**
  It checks if semaphore waiters list is not empty then first thread is taken from the waiters list thread is given lock as Null.
  Also the priority of thread holders are changed appropriately.
- **compare_condition_list (const struct list_elem *a , const struct list_elem *b , void *aux_ UN)**
  it compares 2 elements of condition list .

Also many other small changes have been made to other functions. Also in struct semaphore_elem , a new variable priority is also declared which is used in lock_release.

BSD scheduling
in this a new file is added **fixed point.h** in which all the macros of multipication , addition, division etc are defined when are used further for BSD scheduling .

In thread.h

these two variables are declared
int nice; //to store the nice value for each thread
int recent_cpu; //to store the recent cpu time for each thread

in thread.c

1)thread_tick (void)
This function checks if thread_mlfqs is true then bsd scheduling is implemented.**recent_cpu** time is calculated for the current thread. After every second **load_avg** ,and recent cpu times are calculated for all the threads by traversing through the list of all threads and after every 4 seconds priority is calculated for each thread in the thread list by using the **nice** value and **recent_cpu** time for the thread.
2) thread_set_nice
set nice values of current thread
3) thread_get nice
Return current thread's nice value.
4) cal_priority
made to calulate priority for the current thread.
5) get_recent_cpu
to get recent_cpu time
many other such functions are implemented.

PROJECT 2(USER PROGRAMS)

New Additions in the structure of THREAD in thread.h

- ***struct semaphore wait;*** *// semaphore associated with each thread for process_wait. Initialized with 0 in* ***thread_create.***
- ***int ret_status;*** */*return status for each thread initialized to -5 as default. -1 represents an invalid status which is set if there is an invalid pointer or file or any other wrong access by using exit system call with argument -1.*/*
- ***struct list files;*** *// list of all opened files by the thread*
- ***struct file *self;*** *// it stores the pointer to the image file on the disk*
- ***struct thread *parent;*** *// pointer to the parent process*
- ***struct list children;*** *// list of all children process*
- ***struct list_elem children_elem;*** *//children_elem to be stored in the child list of its parent*
- ***bool exited;*** *//to represents the exit status of the thread. Default value=* ***false***

All these variables are initialised in ***thread_create*** *which is called from* ***process_execute*** *in* ***process.h.***

Newly Created Structures

This structure is used to store all the information about a opened file.

Added in syscall.c

static struct list open_file_list; *//list to store the opened files at any instant*

This structure contain every bit of information of a file.

```
struct file_info
 {
   int fd;              //file descriptor of the file which is assigned starting from 2 and incremented
   struct file *file;// name of the open file
   struct list_elem elem;  //list_elem element to store a given file in the list of open files
   struct list_elem t_elem;  //list_elem element to be added in the process's open file list
 };
```

Newly Created System calls

- ***int system_call_write (int fd, const void *buffer, unsigned length);***

  This is used to write to a file or to a screen. It acquires a lock before writing anything on the console. If the mode is write mode then put_buf function is used to put the output on the screen. Any exception will cause it to release the lock and exit with -1.If everything is fine then the file with fd is written successfully.
- ***int system_call_halt (void);***
  This is used to halt the system. It simply call the function ***shutdown_power_off .***

- ***int system_call_create*** *(const char *file, unsigned initial_size);*
  *This creates a new file. After checking the validity of the arguments passed to it this system call simply creates the file using **filesys_create** function.*
- ***int system_call_open (const char *file);***
  *This opens a new file.After checking the validity of arguments it opens the given file using **file_sys_open.** It also creates an object of **file_info** type and then initializes it with the **file_descriptor,** pushing in the **open file list** and pushing it in the **file list** of the **process** which opens it.*
- ***void system_call_close (int fd);***
  *This closes a already opened file by the currently running process otherwise exits. It first searches the file with the given fd and then close it.It also removes the file information from the **open file list** and the **file list** of its parent.*
- ***int system_call_read (int fd, void *buffer, unsigned size);***
  *This is used to read from the user or read from the file . Before doing anything it first acquires **lock** and releases it before exiting.*
  It first searches for the file with the given fd and reads it if the file is valid otherwise it exits with -1.
- ***int system_call_exec (const char *cmd);***
  *This is used to create a new process. After checking the validity of the arguments it simply calls **process_execute** after acquiring a lock and releases it before finishing the system call.*
- ***int system_call_wait (pid_t pid);***
  *This is wait system call. It simply calls **process_wait** with the provided pid.*
- ***int system_call_filesize (int fd);***
  *This is used to retrieve the size of the opened with the provided **fd** . This returns value returned by **file_length** function for a valid file.*
- ***int system_call_tell (int fd);***
- ***void system_call_seek (int fd, unsigned pos);***
  This system call changes the next byte to be read or written in open file fd to position, expressed in bytes from the beginning of the file

- ***bool system_call_remove (const char *file);***
  *This system call removes the file passed as argument by using **filesys_remove** function if the file exists.*

Newly Created Functions

- ***bool check_address(void *vaddr);***
  *This function checks for the validity of the address.Return true if it is below PHY_BASE else false. It calls **is_user_vadddr function.***
- ***struct file_info *search_file_info (int fd);***
  *This functions returns a pointer to the object of structure file_info corresponding to the passed fd by travesing through the **open_file_list.***
- ***struct file *search_file (int fd);***
  *Return the pointer to a file corresponding to the given fd. It uses **search_file_info** function.*
- ***struct file_info *search_file_in_process (int fd);***
  *This function returns a pointer to the object of the structure **file_info** corresponding to the **fd** passed by traversing through the file list of the currently running thread otherwise returns null.*

Improved/Implemented/Changed functions In Syscall.c

static void syscall_handler (struct intr_frame *f)

*We first retrieve the Syscall number from the **esp** and check its validity and validity of all the arguments in the stack. Then we call the appropriate system call by checking the value of **esp** and then storing the return value in the **eax** pointer of the stack.*

start_process (void *file_name_)

We first load the stack and then fill it with all the arguments, their return addresses, number of arguments etc. We first calculate the length of the command line string and then decrement the ***esp*** *by that value and fill the stack with the command line string. We then align the* ***esp*** *to the multiple of 4. Then we calculate the addresses of the different arguments in an array by using the* ***strtok_r*** *function and fill in the addresses. Then the number of arguments and other things are put into stack and finally the return address in put into the stack.*

In Process.c

tid_t process_execute (const char *file_name)
In this function we have simply passed only the ***file name*** *in the* ***thread_create*** *function instead of the full command line which include file name as well as arguments.*

Int process_wait (tid_t child_tid)
This function first searches for the process with the given child_tid. If the child thread has the default status or an invalid staus or has DYING status then the function sets the return status as -1 and exists otherwise it waits by calling ***sema_down.*** *If the status of the thread is BLOCKED then it is unblocked.*

process_exit (void)
This function first unblocks all the threads waiting on its ***wait*** *semaphore giving them a chance to execute before executing itself**.** Also if its parent exists then it blocks and allows other thread to execute so that we can get its return status(we have to print it) otherwise it will exit and will be destroyed and we will not be able to get its return status.*

In Thread.c

Void thread_exit (void)
In this function the currently exiting thread traverses its children list. If it is BLOCKED then unblock it otherwise make it independent by setting its parent as NULL and remove it from the list of its parent.