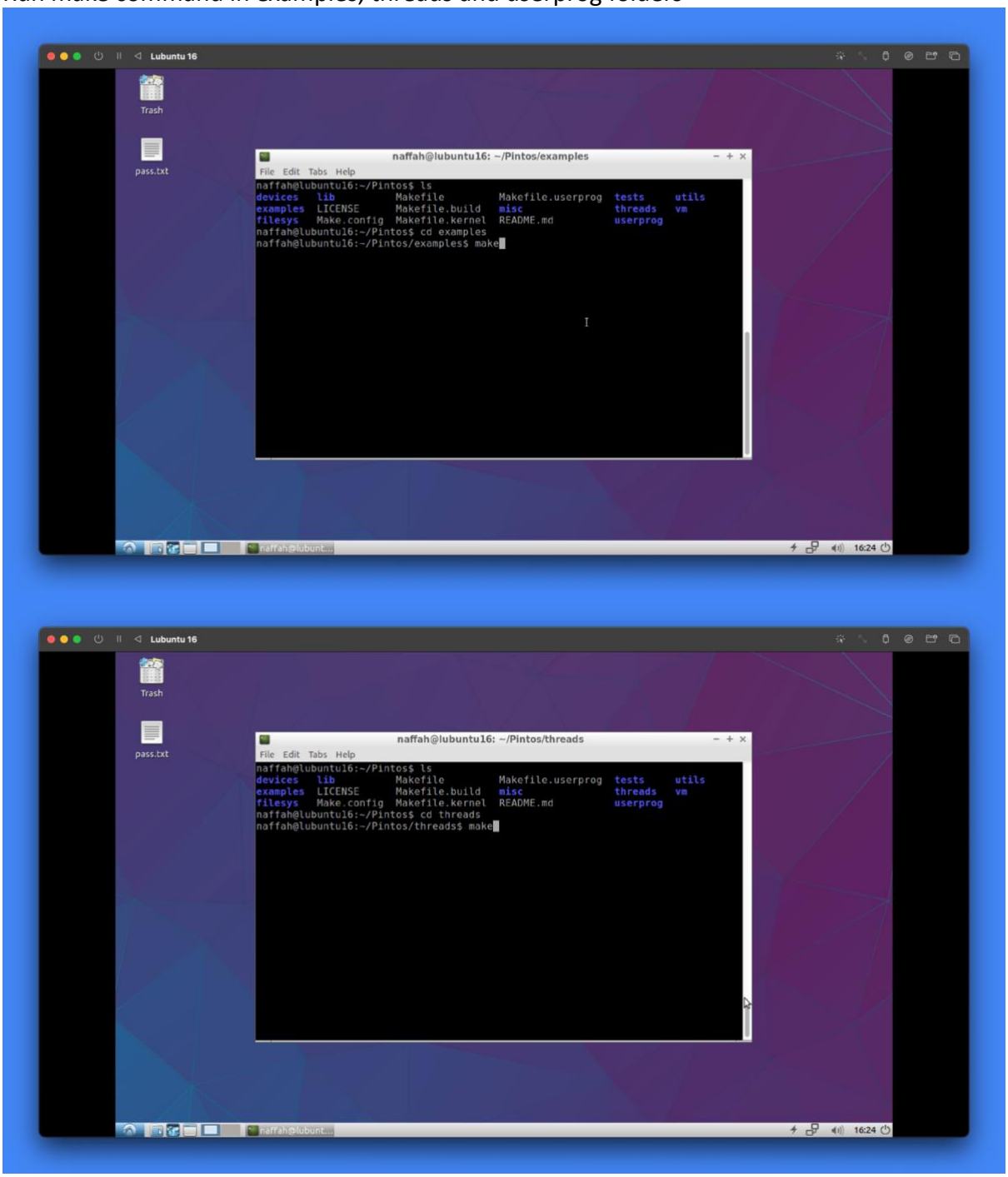
OS Worksheet 3 Documentation

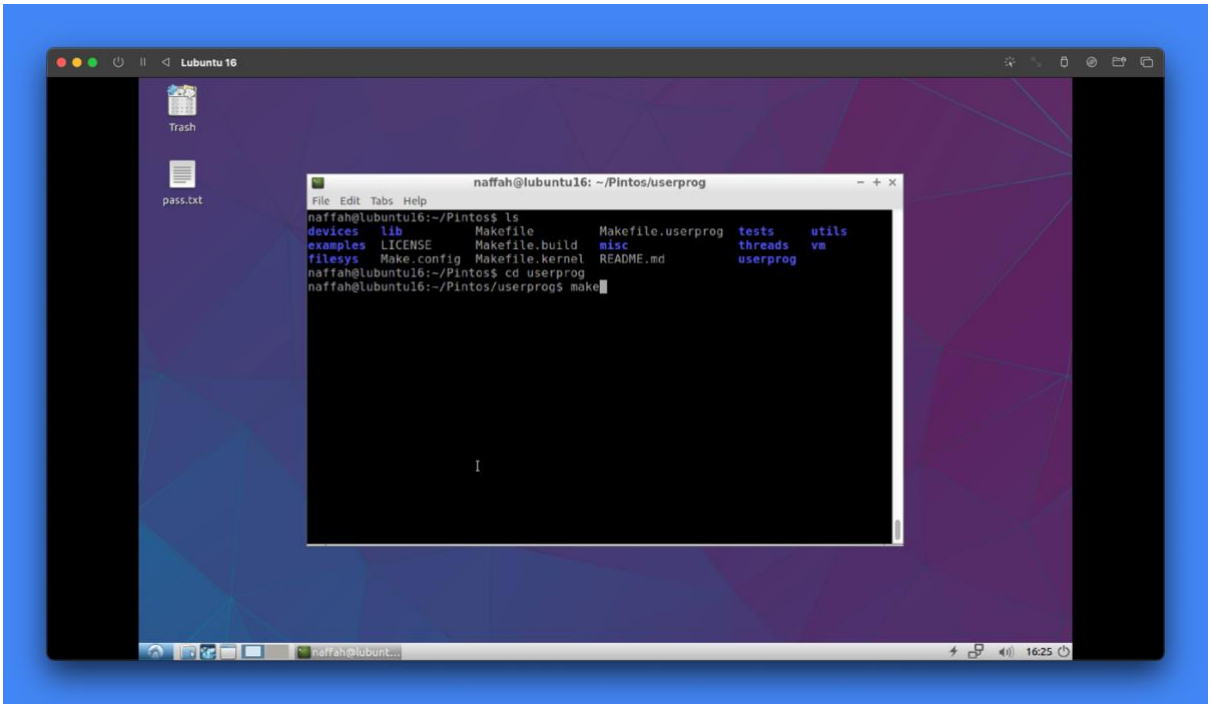
By Naffah Abdulla Rasheed – 19039089

SETUP

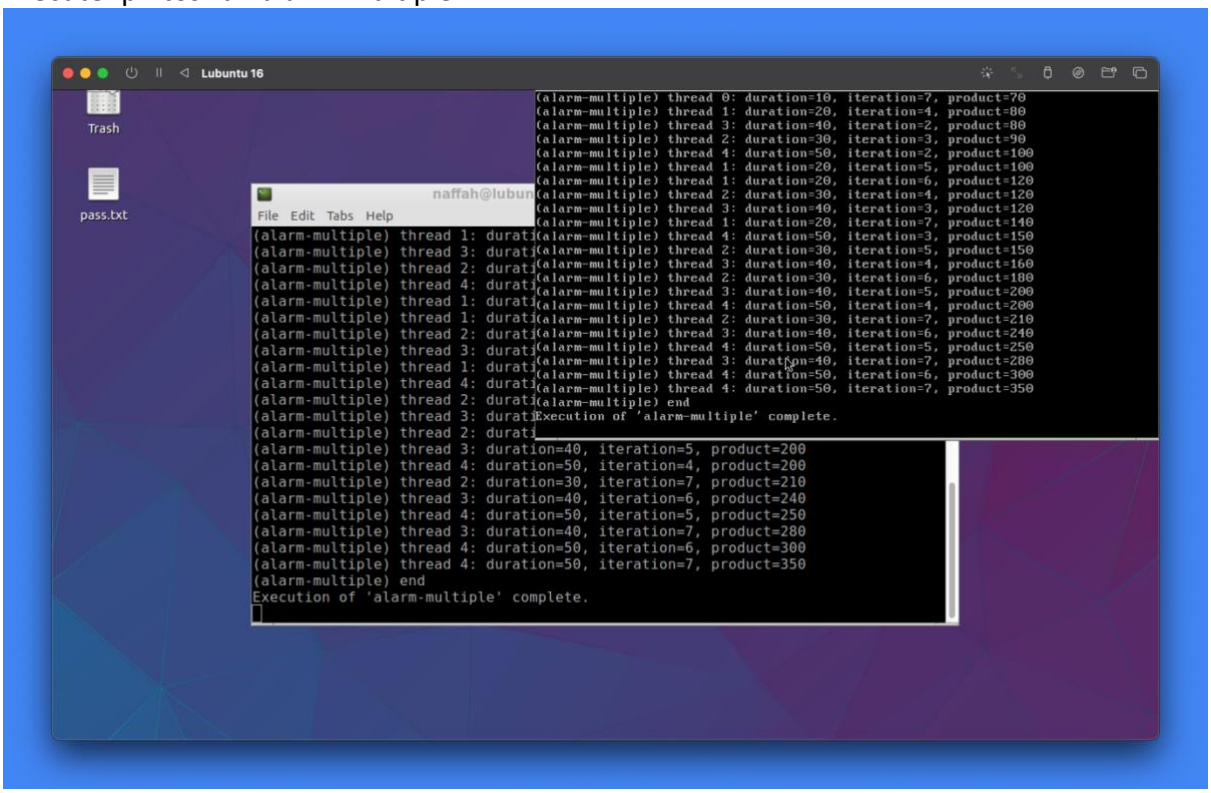
1. Add Pinos/utils to PATH using the following commands:
    a. `export PREFIX="$HOME/"`
    b. `export PATH=$PATH:$PREFIX/Pintos/utils`

2. Run make command in examples, threads and userprog folders

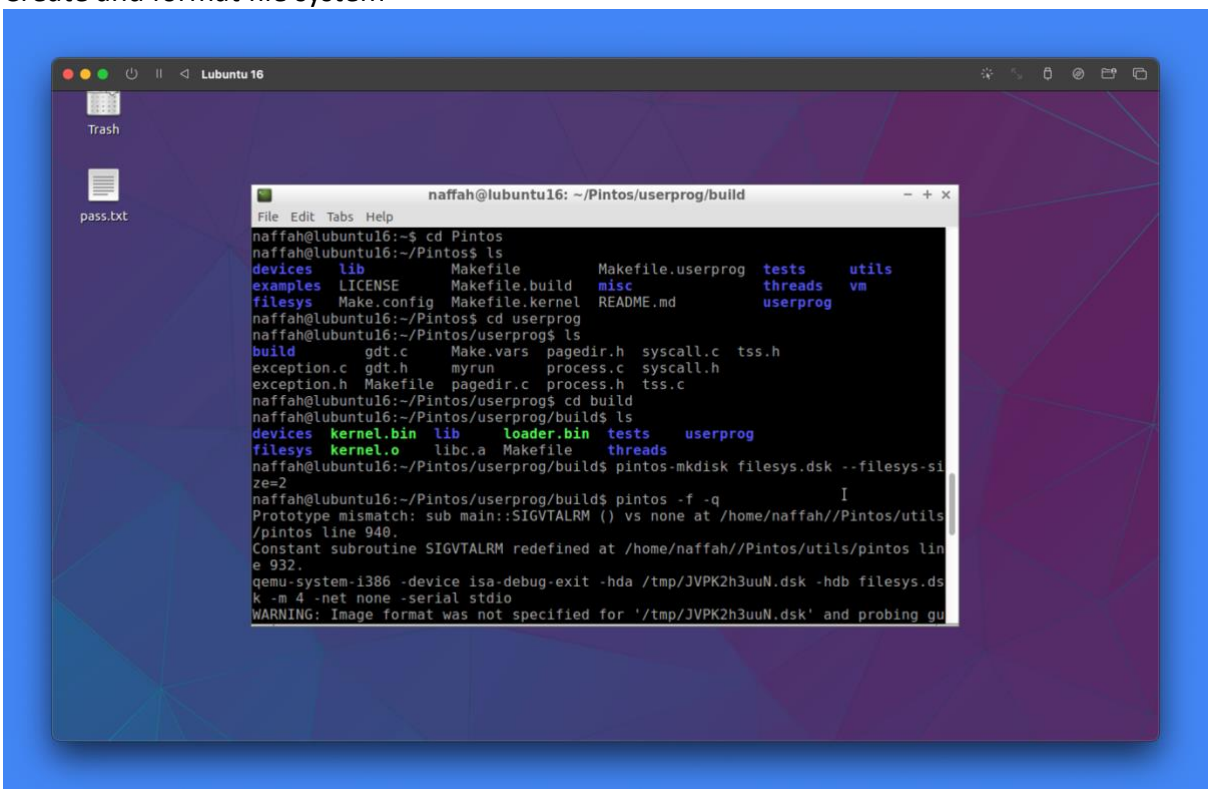




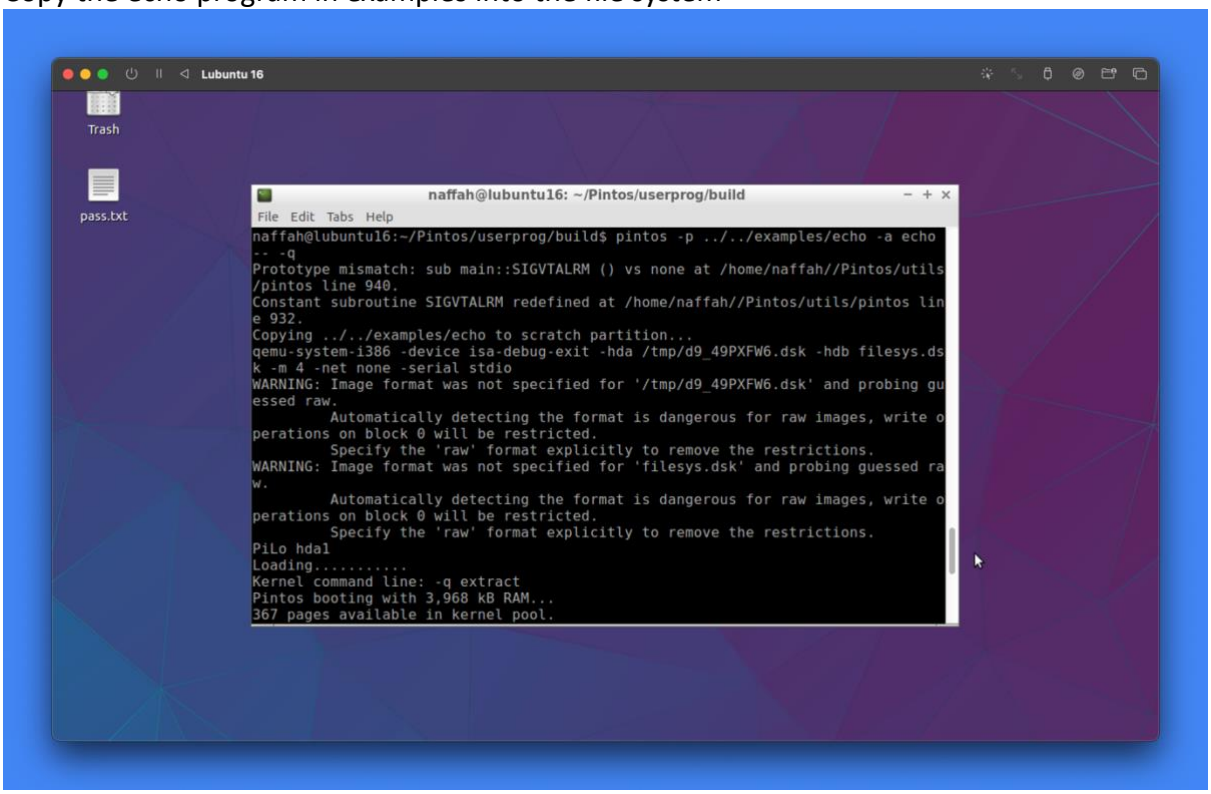
3. Execute `pintos run alarm-multiple`



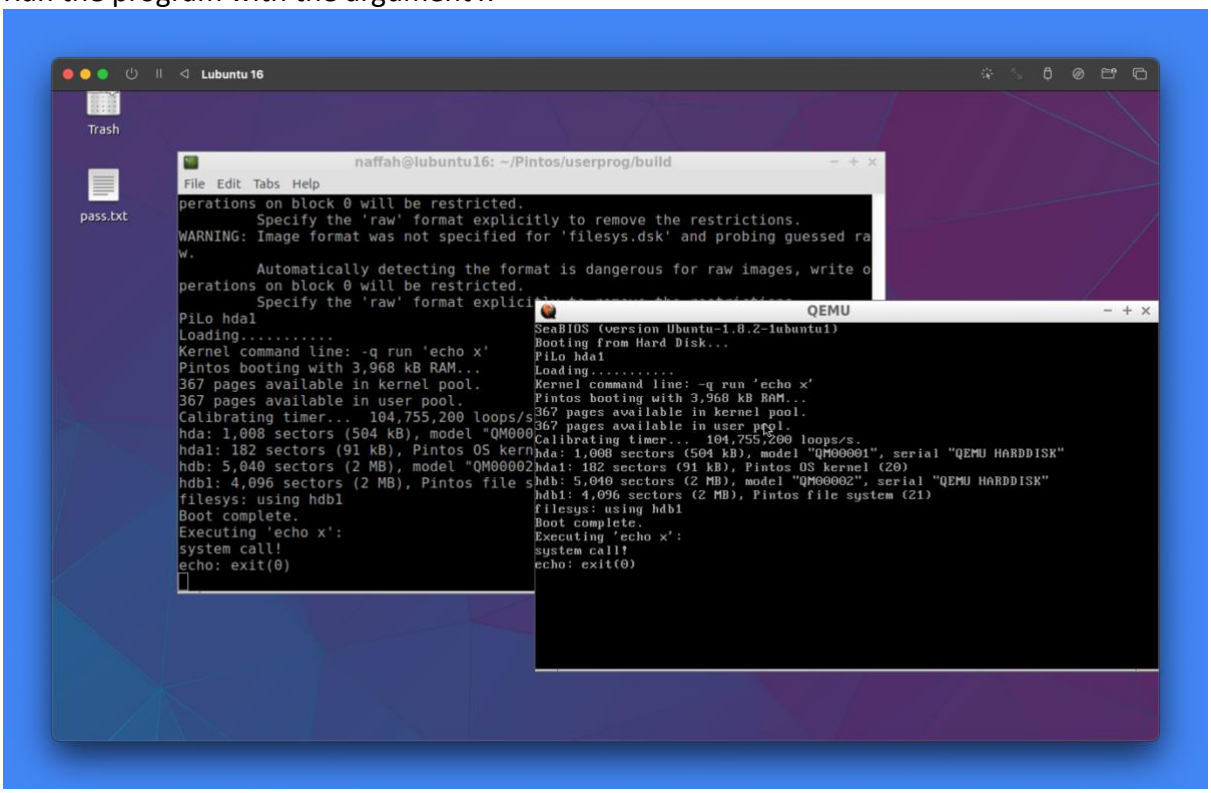
4. Create and format file system



5. Copy the echo program in examples into the file system



6. Run the program with the argument x



CODE MODIFICATION

There are 2 files that I have modified the code in

a. threads/threads.h
b. userprog/process.c

1. Add a field named `exit_status` in the thead struct in threads.h

```
#ifdef USERPROG
    /* Owned by userprog/process.c. */
    uint32_t *pagedir;                  /* Page directory. */
    int exit_status;
#endif
```

2. In order for the process to get the correct file name, remove the arguments in the command stored in file_name variable in process_start() function in process.c, and pass the file name or program name into the load function:

```c
start_process (void *file_name_)
{
  char *file_name = file_name_;
  struct intr_frame if_;
  bool success;

  /* Initialize interrupt frame and load executable. */
  memset (&if_, 0, sizeof if_);
  if_.gs = if_.fs = if_.es = if_.ds = if_.ss = SEL_UDSEG;
  if_.cs = SEL_UCSEG;
  if_.eflags = FLAG_IF | FLAG_MBS;

  // Extract the program name and arguments using strtok_r
  char *save_ptr;
  char *prog_name = strtok_r(file_name, " ", &save_ptr);

  success = load (prog_name, &if_.eip, &if_.esp);

```

3. Extract the file name again in process_execute() function and pass the program name to thread_create() function:

```c
process_execute (const char *file_name)
{
  char *fn_copy;
  tid_t tid;

  /* Make a copy of FILE_NAME.
     Otherwise there's a race between the caller and load(). */
  fn_copy = palloc_get_page (0);
  if (fn_copy == NULL)
    return TID_ERROR;
  strlcpy (fn_copy, file_name, PGSIZE);

  // Extract the program name and arguments using strtok_r
  char *save_ptr;
  char *prog_name = strtok_r(fn_copy, " ", &save_ptr);

  /* Create a new thread to execute FILE_NAME. */
  tid = thread_create (prog_name, PRI_DEFAULT, start_process, fn_copy);

  if (tid == TID_ERROR)
    palloc_free_page (fn_copy);
  return tid;
}
```

4. Finally, print the exit code in process_exit() function along with the file name passed with process execute()

```c
process_exit (void)
{
  struct thread *cur = thread_current ();
  uint32_t *pd;

  /* Destroy the current process's page directory and switch back
     to the kernel-only page directory. */
  pd = cur->pagedir;    struct "thread" has no field "pagedir"
  if (pd != NULL)
    {
      /* Correct ordering here is crucial.  We must set
         cur->pagedir to NULL before switching page directories,
         so that a timer interrupt can't switch back to the
         process page directory.  We must activate the base page
         directory before destroying the process's page
         directory, or our active page directory will be one
         that's been freed (and cleared). */
      cur->pagedir = NULL;    struct "thread" has no field "pagedir"
      pagedir_activate (NULL);
      pagedir_destroy (pd);
    }

  printf ("%s: exit(%d)\n", cur->name, cur->exit_status);    struct "th
```

5. If pagefault occurred change *esp = PHYS_BASE; to *esp = PHYS_BASE - 12; in setup_stack() in process.c

```c
setup_stack (void **esp)
{
  uint8_t *kpage;
  bool success = false;

  kpage = palloc_get_page (PAL_USER | PAL_ZERO);
  if (kpage != NULL)
    {
      success = install_page (((uint8_t *) PHYS_BASE) - PGSIZE, kpage, true);
      if (success) {
        *esp = PHYS_BASE - 12;
      } else
        palloc_free_page (kpage);
    }
  return success;
}
```