CS 450

Module R6

Next Week

- R5 is due next Friday
  - No documentation, no source code due
  - R5 is stand-alone, so I will not be checking errors from previous modules. But in R6 everything needs to work.

Note: Dates marked "???" will be determined

R6

- Due ??? at the time of your group's oral exam
- Documentation- due at the oral exam:
  - Programmer's Manual
  - User's Manual (but not the temporary commands manual)
  - Group Portfolio
  - A complete copy of your source code
    - I will check that you have included the comments that are required and you will lose points if they are missing
    - Turn in these documents- bound in some fashion- either a three ring binder or whatever method you want, but do not just staple them, for this module **they must be bound.**
  - Also, you must turn in a CD with all of your source code (including help files), as well as your programmer's manual and user's manual. Also, please include a README file with the following information on the CD:
    - Group number and group member names and semester
    - Compiler and version number you used
    - Operating System used for development
    - If you used a DOS emulator (DosBox) or not
    - Any other information that might be needed to run your MPX project
- You will need to use the serial port emulator, or two computers physically connected, to demonstrate R6 just like R5.

Oral Exams

- Oral exams will take place during dead week.
- On ???, we will schedule the oral exams- make sure one group member comes so you can schedule your oral exam.
- **If no one from your group shows up on ??? to schedule, your group's oral exam takes place on the first available day during dead week. NO EXCEPTIONS.**
- All group members must attend the Oral Exam- if you do not attend, you will FAIL the class.
- The oral exam will last approximately one hour and will be comprised of R6 demonstration and questions from Dr. Mooney and myself regarding the MPX project. You are expected to know everything about every portion of the project, including how you implemented specific things, and why specific functions are needed even if you didn't work on that particular part of MPX.
- You will find it VERY useful to bring a complete copy of your source code for each group member to the oral exam. It is also helpful if you include a table of contents and tabs for each module so you can find specific functions quickly to answer questions.

R6: Overview

- We need to implement “continuous dispatch”
  - Comhan will be loaded as a process.
  - An IDLE process will be introduced, so there is always something in the ready queue and ready to dispatch.
- We will also need to provide more support for I/O Requests
  - An I/O scheduler and completion handler will be added to handle I/O events.
  - Support for the terminal driver provided in the support software will be added.

Commands

- At this point, only the permanent commands should be in your system- all temporary commands should be disabled!
- Commands that are temporary and should be removed:
  - Block
  - Unblock
  - Loading the R3 test processes
  - Dispatch
  - Create-PCB
  - Delete-PCB

- Commands that should be available in R6
  - Set/get date
  - Display MPX files
  - Help
  - Load
  - Quit/Exit
  - Suspend*
  - Resume*
  - Terminate*
  - Set Priority*
  - Show all/blocked/ready/specific PCB
  - Version
  - Any commands relating to extra credit

*Do not allow users to suspend, resume, terminate, or change the priority of SYSTEM processes.

Comhan and IDLE

- In order to obtain “continuous” dispatch- we will need to load comhan and an idle process into the ready queue when MPX starts.
- Comhan
  - Whatever your procedure name is that handles your commands (the one that prompts the user for a command and receives it) needs to be installed as a process in MAIN.
    - Do this the same way you loaded the test processes in R3
    - Give Comhan the **highest** priority, place it in the ready, not suspended state, insert it into the ready queue.
    - You may find it necessary to give Comhan a larger stack space than other processes- minimum recommended stack size for Comhan is 4K.
  - When comhan gets the “exit” command- it should remove all processes from the queues, including IDLE and itself, so that when dispatch is run again, it detects an empty ready queue and returns to main.
- IDLE- available on my website
  - Load idle- use your “load” function from R4
    - Give IDLE the LOWEST priority
    - Make sure IDLE is designated a SYSTEM process
    - Resume it and put it in the ready queue
  - IDLE doesn’t do anything, it just ensures there is always a process in the ready queue and thus dispatcher won’t quit until we want it to.

I/O Event Handling

- The basic idea here is this: while an I/O device is carrying out an I/O request and a process is waiting on the I/O request, another process should be allowed to execute (have control of the CPU).
- We will need to add functionality that allows us to queue I/O requests, by device, and start an I/O operation and run another process.
- We will also need to implement a way to handle the completion of an I/O operation.

Handling I/O Events

- Devices
  - Com Port- driver written by you in R5
  - Terminal- driver supplied on my website
- Data Structures
  - IO Control Block (IOCB)- holds information about an I/O device
  - IO Request Descriptor (IOD)- holds information about an I/O request
- I/O Scheduler
  - Processes I/O requests, either by processing immediately if the device is free or by placing it on the device's waiting queue if the device is busy.
  - It is most convenient to put your I/O scheduler and all related data structures in your R3 files.
- Sys_call
  - Each time sys_call is invoked, it will check to see if any I/O events have been completed- if so:
    - COP must be switched from the blocked to ready state and moved from the blocked to ready queue
    - Active IOCB (device) must be cleared to signal that no request is active for the device
    - Search the waiting queue for another process waiting to use the device that was just freed. If one is found, the I/O scheduler is called to start the I/O.

I/O Request Descriptors (IOD)

- We need to keep track of certain information for every active or pending I/O request
- Two options for where to put the IOD's
  - An MPX process can only have 1 pending I/O request at a time, so it is possible to embed an IOD in each PCB. However, this is not very convenient when it comes time to find the next I/O operation to be run.
  - Create a "waiting queue" of IOD's for each device- much more convenient, and I will assume you use this option in the remaining slides.
- Suggested information to keep track of:
  - Name of the process (PCB) requesting the I/O operation (char[])
  - Pointer to the PCB requesting the I/O operation
  - Request type: what kind of I/O request (i.e. IDLE, READ, WRITE, CLEAR) (int)
  - Location of the transfer buffer (char*)
  - Pointer to the count variable of the buffer (int*)
  - Pointer to the next IOD in the waiting queue
  - Note that the request type, transfer buffer, and count variable refer to the op_code, buffer_address, and count_address in the parameter* you use in R3

I/O Control Block (IOCB)

- Each device needs a control structure which will keep track of the waiting queue and current status
- Suggested information you need to keep track of in the IOCB:
  - event_flag: int- will hold the current event_flag for the device
  - count: int- number of IOD's in the queue
  - head: iod*- pointer to the first IOD in the waiting queue
  - tail: iod*- pointer to the last IOD in the waiting queue
- You will maintain the waiting queues in first in, first out order.

I/O Operations

- You will need to download and link with your project “TRMDRIVE.C” and “TRMDRIVE.H”- these are the functions for the TERMINAL driver. You can open the files to see the parameters the functions take. Make sure to include TRMDRIVE.H and link TRMDRIVE.C. They will be available on my website.
- The following table illustrates what functions to call for which I/O operations- the COM_PORT functions come from your R5, check TRMDRIVE.C and .H for more specifics on the TERMINAL functions

Device	OPEN	CLOSE	READ	WRITE	CLEAR	GOTOXY
COM_PORT	com_open	com_close	com_read	com_write	N/A	N/A
TERMINAL	trm_open	trm_close	trm_read	trm_write	trm_clear	trm_gotoxy

Processing an I/O Request

- In both sys_call and the I/O Scheduler, it will be necessary to process an I/O Request. Here is the basic outline of how you would do that:
  - Decode the request (from the IOD->request) for the operation at the head of the queue, it will either be READ, WRITE, CLEAR, GOTOXY
  - You will now need to call the appropriate function to handle the request:
    - The parameters are the same if you are calling a read or write (trm_read, for example):
      - IO_FUNCTION(com_iocb.head->buf_addr, com_iocb.head->count_addr)
    - Call the appropriate operation, using the syntax above, to handle the request. Check the chart on the I/O operations slide for information on which function to call.
      - Note that CLEAR and GOTOXY for the TERMINAL device require different parameters than those listed above.

I/O Scheduler

- The I/O scheduler is a function you will write that will process a new I/O request.
- You will recall that in R3 you used a pointer to “parameters” that were pushed on the stack when an interrupt was generated- we will need to access those parameters again.
- Because of this, it makes your coding very convenient if you place the I/O Scheduler in your R3.C file, and place the IOD and IOCB structs in R3.H
- It also makes it very convenient if you make your parameter* (that you used in R3) global, if you haven’t already.
- If you make the parameter* global, you will not need to pass any parameters to your IO Scheduler.
- In short, the I/O scheduler will create a new IOD for the I/O request, and either run it if the device is ready, or place it in the waiting queue to be run later.

I/O Scheduler: Outline

- Check which device is being requested (stored in the device_id of the parameters)
- Create an IOD for the new request (allocate space using sys_alloc_mem)
  - Set the PCB* in the IOD to COP, and set the name-> The COP is the process making the I/O request, so you need to save that in the IOD
  - Set the buffer address to the parameter* buf_addr, and count address to the parameter* count_addr, and the request to the parameter* op_code
- If the waiting queue for that device is empty (count = 0)
  - Insert the new IOD into the waiting queue as the head and tail, set count = 1.
  - Process the request immediately- refer to the slide on processing I/O requests on how to do this
- If waiting queue for that device is not empty
  - Insert the new IOD into the waiting queue at the tail
- Block the process and return.

I/O Completion

- Every time sys_call is invoked, it will need to check if any I/O operations have been completed.
- Sys_call will do this by checking the event flags in each IOCB- if it is set, some “completion” steps need to be performed

Sys_call modifications

- Call trm_getc()- this will flush the DOS keyboard buffer into the MPX buffer
- For each I/O device (COM_PORT and TERMINAL):
  - Check if the event_flag is set (=1)
  - If the flag is set
    - Clear the IOCB's event_flag (set = 0)
    - Remove the IOD at the head of the IOCB's waiting queue and **free the IOD's memory**
    - Unblock the process who had requested the IOD you just freed, this should put it in the ready queue. The process that requested the IOD is maintained as a field in the IOD
    - Process the next I/O request for that device
- Suggested to perform these steps after you switch to the system (temporary) stack, and before you reset COP's stack top pointer.
- At the end of sys_call, where you currently check the op_code to see if it is IDLE or EXIT, you need to add checks for READ, WRITE, CLEAR, GOTOXY- if the op_code is any of these values, call your IO Scheduler to handle the interrupt.

I/O Initialization

- You need to create and initialize the IOCB's for the terminal and com_port devices
- You need to open the terminal and com_port devices
  - trm_open(POINTER TO THE IOCB EVENT FLAG);
  - com_open(POINTER TO THE IOCB EVENT FLAG, baud_rate);
    - Set the baud rate = 1200
  - By passing a pointer to the event_flag, this allows the terminal and com interrupt handlers to directly modify the event_flag.

I/O Cleanup

- When MPX is terminated
  - Close the devices by calling trm_close and com_close
  - Clear the waiting queues, free the memory of any IOD's in the queue
  - Free the memory for the IOCB for each device

Main

Your main should do the following now-

1. sys_init(MODULE_F)
2. Call Init functions- you might have already written init functions in R2 and R3 that do the following:
   - sys_set_vec needs called
   - initialize DCBs, IOCBs
   - initialize PCB ready/blocked queues
3. Open device drivers
4. Install command handler as a process
5. Load IDLE
6. Call dispatcher- this should run your comhan
7. Close device drivers
8. Other cleanup- you might have cleanup functions in R2 and R3 to call
   - Make sure the queues get cleared and all allocated memory is de-allocated
9. sys_exit()

Test Processes

- Available on my website
- Load these using your load function to test your project.
- **CPUTERM**. A process which repeatedly displays a message line on the display screen. This process is CPU bound; it waits for a while in a loop between messages, consuming processor time. It runs continuously until terminated.
- **CPUCOM**. Similar to CPUTERM, but this process outputs its messages to the communication port. We assume that a standard terminal, or a separate PC or workstation running terminal emulation software, is connected to this port.
- **IOTERM**. Similar to CPUTERM, but this process includes no delay loop. It attempts to continuously output messages, making a new request as soon as the last one has been completed.
- **IOCOM**. Similar to IOTERM, but this process outputs its messages to the communications port.
- **IOTRM25**. Similar to IOTERM; however, this process displays its normal message exactly 25 times, then requests termination. A special message is displayed before the termination request. If the process is dispatched after its termination request, an error message is displayed, and the process restarts.
- **IOCOM25**. Similar to IOTRM25, but this process outputs all of its messages to the serial port.
- **IOMULTI:** Takes input from the com port and prints it to the screen.

Preliminary Code

- On ???, you must turn in “preliminary” code for R6. I want to see that you are starting on the code- have coded the section where Comhan gets loaded as a process and you are making progress towards the I/O scheduler. Your code does not have to work or even compile.
- My intention is to force you to start working on R6 early and get the code written so you have 2 weeks to dedicate just to debugging.
- Code is due at the beginning of the project session on ???. You may print out the appropriate sections of code and hand it in or email me a copy of your source code.
- If you turn in the preliminary code on:
  - ??? you lose no points
  - ??? you lose 7 point
  - ??? you lose 14 points
  - If you fail to turn in by ???, you lose 20 points.

Tips

- You cannot use any I/O device before you call trm_open and com_open- this includes using printf.
- Once you change sys_init to MODULE_F, MPX will only work if your I/O scheduler works!
- Remember that you have up to 3 extra credit points available to you- some of these are very simple to implement.
  - Change prompt, history, aliasing, accepting commands from a script/batch file
- Your programmer's manual should have documentation for **every** function in your code and every structure in your code
- Your user's manual should only have documentation for all permanent commands
- Documentation will be graded more thoroughly than previous attempts
- All functionality of MPX will be tested again- so make sure everything works!
- Make sure to thoroughly test your project- crashes in particular will reflect poorly in your grade.
- Make sure you have descriptive error messages displayed anytime there is an error.
- Make sure that help is there for all permanent commands.
- Make sure you implement paging for help, show, and anywhere else it is appropriate.
- The R6 project manual will be available on my website- as always, reading it will make this module easier to complete
- For Module R6, do not use anything from Dr. Mooney's site- it is not in sync with the R6 you are doing this semester.