Agenda

• High level overview of steps to finish lab 2 from the perspective of the cookbook starting with step 5 and onwards
• What will I not go over:
  • Step 1 – 4 and the timer was covered by Sierra
• Q&A
KOS Address vs User Address

• KOS VIEW of memory:
  • The green part is allocated to a user

• USER VIEW of memory:
  • The user will always think their address space starts at 0
  • If the user passes in address 1, then it is really address 4
  • Any address you give to the user will be in their address space
  • EVERY address that the user program uses must be a user address
User Base and User Limit

- User_Base is the KOS address where the user’s memory partition starts
- User_Limit is the size of the memory that the process is given
- Each PCB keeps track of what partition of memory it is assigned

- These two processes both get 50 bytes of memory but start at different bases
- Before calling run_user_code, set the global User_Base and User_Limit to the process that is about to be run
execve

• Recommend to factor out part of initialize_user_process()
• perform_execve(pcb_block, fn, argv)
  • Load user program, set the registers, and place argv on the stack
• Why separate this?
  • You can reuse this code when you are implementing execve()
• When the user program calls execve(), you need to catch the exception and then handle it
execve(const char *pathname, char *const argv[], char *const envp[])

• You must malloc some new space and place the pathname and argv into memory

• **Why?**

Memory:

<table>
<thead>
<tr>
<th></th>
<th>pathname</th>
<th>argv[0]</th>
<th>argv[1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>New stuff</td>
<td>New stuff</td>
<td>New stuff</td>
<td></td>
</tr>
</tbody>
</table>

• When you call load_user_program, you will load the new program into memory and overwrite what is currently there

• Now all your parameters are gone. You have nothing to place on the stack

• Instead: Save them on the heap first
Process IDs

- Every process needs a unique ID
- You need to ensure that you do not reuse IDs
- The cookbook says to use a red-black tree, but you can use anything, as long as no 2 processes ever share an ID
  - "/cs/faculty/rich/cs170/include/jrb.h" ← RB tree header file
- The code does not have to be fast. As long as a the test does not take longer than 60 seconds
fork()

• Create a new PCB and allocate it a separate partition of memory
• The new pcb should have an exact copy of all the old PCB registers
• The new pcb should have a different base and limit
• After fork() finishes, there should now be one additional PCB on the readyqueue
• kt_fork(finish_fork, new_pcb)
  • syscall_return will increment the current pcb’s program counter and return the new_pcb ID in register 2
  • However, we also want to increment the new_pcb program counter and it should return 0 because it is the child
  • Spawn a new thread to handle the new pcb
exit(int status)

• Do not kill the OS anymore with a SYS_HALT
• Release the memory partition so that other programs may use it
• Save the exit value in the PCB
  • You can create another variable within each PCB block for this
• Do not deallocate the PCB. It will just exist in memory as a zombie until it is cleaned up by wait()
• Do not put the PCB back on the readyqueue. It will never be run again
wait()

- Each PCB needs to be modified to keep track of who its parent is
- Use a pointer that points to the parent block
- The first process should be the child of an Init PCB
  - Init never gets run. It only exists so that the first pcb has a parent
wait()

• Each PCB block needs to keep track of all the processes that are waiting on it
• When a child calls exit(), it must be added to its parent’s waiting list
• wait() will clean up one child or will block if no children are ready to be cleaned
wait()

- When a process dies, its children become the children of Init
- PCB block needs to be modified again to keep track of all its active children
- When fork() is called, the new child PCB must be added to its parent’s active children list
- When exit() is called, all active children must be moved to be Init’s active children
- Multiple ways to cleanup the children of Init: See cookbook
Q & A