Schedule

• Announcements
  – Send me email with teams and team name
  – Demo times (4/26, 5/17, 6/7 from 3-7)
  – Anonymous Feedback
    • http://cs.ucsb.edu/~bboe/p/suggestion

• Project 1
  – Simple Shell (35 minutes)
  – Lottery Scheduler (15 minutes)
Project 1 - Two Parts

• Simple Shell
  – Read input from standard in
  – Handle special cases with >, <, |, &
  – All shell output to stdout
  – No debugging output

• Minix Lottery Scheduling
  – Piggyback lottery scheduler for user processes on existing priority scheduler
  – Add system call to change number of tickets for a process
What is a shell?

- Control program execution
- Manage program input and output
- Control working directory
- Switch between foreground and background processes
System Calls You Will Need

• fork – copy current process including all file descriptors
• exec – replace running process with code from program (file descriptors persist)
• waitpid – current process waits until pid terminates
• pipe – create memory mapped file
• dup/dup2 – update process file descriptor numbers
Great C Resource

• Opengroup.org
• Google search to find command:
  – fork site:opengroup.org
pid_t fork(void)

int pid;
switch (pid = fork()) {
    case 0:
        /* child process */ break;
    case -1:
        /* error */ break;
    default:
        /* parent process when pid > 0 */
}
int execvp(const char *file, char *const argv[]);

char *argv[] = {"ls", "-la", "/tmp", NULL}

if (execvp(argv[0], argv))
    /* exec failed, maybe file not found */
else
    /* guaranteed to never enter here */
pid_t waitpid(pid_t pid, int *stat_loc, int options);

pid_t child_pid;
int status;
if ((child_pid = fork() != 0) {
    waitpid(child_pid, &status, 0);
    printf("%d\n", status); /* should be 1 */
} else {
    exit(1);
}
int pipe(int *fildes[2]);

int fildes[2]; char buf[BUFSIZ];
if (pipe(fildes)) { /* error */ }
if (fork()) {
    close(fildes[0]); /* close read end */
    write(fildes[1], "foobar\n", 7);
} else {
    close(fildes[1]); /* close write end */
    read(fildes[0], buf, 7); /* reads foobar */
}
int dup2(int fildes, int fildes2);

/* redirect stdout to a file */
int fp;
fp = open("/tmp/somefile", 'w'); /* 3 */
close(STDOUT_FILENO); /* close 0 */
dup2(fp, STDOUT_FILENO); /* clone 3 to 0 */
close(fp); /* close 3 */
Parsing Commands

• Command input represents a grammar
  – Begin -> command (‘<‘ file)? (‘>’ file)? ‘&’?
  – Begin -> command (‘<‘ file)? ‘|’ Extended
  – Extended -> command (‘>’ file)? ‘&’?
  – Extended -> command ‘|’ Extended

• Must parse the commands properly and create the execution chain
Process Tree Creation Questions

• How do we launch a single process and have the shell wait?
  – What about I/O redirection?

• How do we launch two processes with a pipe between them?
  – Which process does the shell wait on?
  – What file descriptors does each process inherit?
Current Minix Scheduler

• Priority scheduler
  – 1. Kernel tasks (system task / clock task)
  – 2. Device drivers
  – 3. Server processes
  – 4. User processes
  – Last. Idle process

• Implemented with 16 queues

• Highest priority process in ‘ready’ state is run
Running Processes

• Each process has a quanta (total time to run)
• Higher priority queues may provide more quanta
• Processes run until either
  – They give up the CPU when making a system call such as IO (return to the head of their queue when ‘ready’ again
  – Their quanta is exhausted (return to end of current queue, higher queue, or lower queue depending)
Switching Queues

- If there are no other “ready” processes when a process exhausts its entire quanta twice the process moves up to a higher priority queue.
- If there are other “ready” processes when a process exhausts its entire quanta twice the process moves down to a lower priority queue.
- Processes can request a change via the nice system call.
Priority Scheduler Questions

• Which is given priority IO bound or CPU bound?

• Can high priority processes starve low priority processes?

• What happens if a device driver or server (high priority) enters a CPU bound infinite loop?
Lottery Scheduler

- Each (user) process is given 5 tickets to start
- At each scheduling decision:
  - chose a random number between 0 and the total number of assigned tickets - 1
  - schedule process “holding” that ticket
- Processes can modify their priority via setpriority(ntickets) (max 100 tickets)
Integration

- Keep it simple
- Only user processes are required to use the lottery scheduler
- Do not need to worry about breaking the nice system call for user processes
- Do not need to worry about handling the setpriority system call for kernel/device/server processes
To consider

• How do we find the process that is elected?
• Incrementally test small changes to ensure you haven’t broke the scheduler
• Write minix user programs to test functionality
  – Equivalent processes running concurrently should complete in the same time
  – If process A has twice the priority of process B it should complete in approximately half the time
Good luck!

• Questions?