## Operating Systems Project 2

|  |  |  |  |
| --- | --- | --- | --- |
| Participants name – Part A | Code Section | Report Section | Documentation Section |
| Cody Krause | Worked together with Jake Karas and Eric Weakley to learn and modify the code along with trying to make sense of the errors that were occurring throughout the process to correct them. | Created and wrote the report for the project, including filling in the rules and writing the descriptions for the modifications to each of the files mentioned | Created the discord server for the project group and set up the time for the meetings. |
| Eric Weakley | Worked together with Cody Krause and Jake Karas to learn and modify the code along with trying to make sense of the errors that were occurring throughout the process to correct them. Managed to figure out the main issue with the scheduler and finished it. | Reviewed the report and reported and added modifications needed for the report | Kept the team updated on any of the changes, errors that have been found with code and correction made |
| Jake Karas | Worked together with Cody Krause and Eric Weakley to learn and modify the code along with trying to make sense of the errors that were occurring throughout the process to correct them. | Reviewed the report and reported and added modifications needed for the report | Kept the group up to data on his visits to the TA to try and grasp a better understanding of the project |
| Participants name – Part B | Code Section | Report Section | Documentation Section |
| Cody Krause | Worked together with Jake Karas and Eric Weakley to learn and modify the code along with trying to make sense of the errors that were occurring throughout the process to correct them. | Created and wrote the report for the project, including filling in the rules and writing the descriptions for the modifications to each of the files mentioned | Created the discord server for the project group and set up the time for the meetings. |
| Eric Weakley | Worked together with Cody Krause and Jake Karas to learn and modify the code along with trying to make sense of the errors that were occurring throughout the process to correct them. Managed to figure out the main issue with the scheduler and finished it. | Reviewed the report and reported and added modifications needed for the report | Kept the team updated on any of the changes, errors that have been found with code and correction made |
| Jake Karas | Worked together with Cody Krause and Eric Weakley to learn and modify the code along with trying to make sense of the errors that were occurring throughout the process to correct them. | Reviewed the report and reported and added modifications needed for the report | Kept the group up to data on his visits to the TA to try and grasp a better understanding of the project |

|  |  |
| --- | --- |
| File | Description |
| Include/syscall.h | All that needs to be done is adding two new system calls for the file as shown in the image below for both part a and part b |
| Include/pstat.h | We had to create a new file pstat in the include folder for the schedulers so that we can collect data from them. |
| Kernel/defs.h | For both a and b, we need to modify this file so to have the struct for pstat and that the system calls are part of proc.c |
| Kernel/sysproc.c | This is where we have the functions for the systems calls we created earlier. Here is Part A’s code      Part B |
| Kernel/proc.h | All we add is number of tickets and number of ticks    int tickets;    // add numticks here    int ticks; |
| Kernel/proc.c | We need to include pstat.h at the beginning of the file and need to add p->tickets and p->ticks  p-> tickets is used for our priority settings in part b hence we make it 50 by default in part b  In both, we write np->tickets = proc-> tickets  For the lottery we need to add within the for loop  int totalTickets =-1;      int numOfWinners =1;      int accessNum=1;  and we need to add the random number generator  lotteryWinner = rand() % (totalTickets + 1);  we need to define rand in the file as well  #define SCHRAND\_MAX ((1U << 31) - 1)  #define SCHRAND\_MULT 214013  #define SCHRAND\_CONST 2531011  //Pseudo-random number generator made from a linear congruential generator.  //Based on code and constants found at:  //https://rosettacode.org/wiki/Linear\_congruential\_generator  int rseed = 707606505;  int rand(void)  {    return rseed = (rseed \* SCHRAND\_MULT + SCHRAND\_CONST) % SCHRAND\_MAX;  }  For part b, we add to the scheduler  for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){        if(p->state != RUNNABLE)          continue;        if(p->tickets > totalTickets) {          continue;        }        if(prevTicketWinner == totalTickets && prevAccessed < numOfWinners && accessNum < prevAccessed + 1) {          accessNum++;          continue;        }        p->ticks++;        prevTicketWinner=totalTickets;        prevAccessed=accessNum;  For both we add the assigntickets function here as well  int assigntickets(int passTickets)  {    //make validation here, if you want    if(passTickets > 0) {      proc->tickets  = passTickets;      return 0;    } else {      return -1;    }  }  And the getpinfo function  int getpinfo(struct pstat\* LaTable)  //create a pointer able to point to object of the tpe pstat  {    if(LaTable == NULL) {      return -1;    }    struct proc \*p;   //Create a pointer able to point to objects of the type proc (process)    int i = 0; // used to iterate througt the slots of the arrays in pstat    acquire(&ptable.lock);  //lock the ptable (array containing the process)    for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){  // use p to iterate throght the ptable      if(p->state == ZOMBIE || p->state == EMBRYO){  //check the state of a process, if it is different of ZOMBY and EMBRIO        continue;      }      if(p->state == UNUSED){        LaTable->inuse[i] = 0;  //check the name of the arrays in pstat.      }      else{        LaTable->inuse[i] = 1;      }      LaTable->pid[i] = p->pid; //with the pid of the process p->      LaTable->tickets[i] = p->tickets; //with the number of ti      LaTable->ticks[i] = p->ticks; //with the number of time the process has runned in the cpu      i++;    }    release(&ptable.lock);    return 0;  } |
| Kernel/syscall.c | We are just adding to the array our new syscalls  [SYS\_assigntickets] sys\_assigntickets,  [SYS\_getpinfo] sys\_getpinfo, |
| Kernel/sysfunc.h | We are just adding to the sysfunc handlers for our system cals  int sys\_assigntickets(void);  int sys\_getpinfo(void); |
| User/tickettest.c | Our new file to test the scheduler  include "../include/types.h"  #include "../include/stat.h"  #include "../include/pstat.h"  #include "../user/user.h"  int main() {      int rc;      int i;      double p = 0;      for(i = 1;i<=5;i++) {          rc=fork();          if(rc<0) {              printf(1,"Fail state: %d\n", getpid());              return -1;          } else if(rc == 0) {              assigntickets(i\*10);              printf(1,"Created %d with %d tickets\n", getpid(),i\*10);              for(;;) {                  p++;              }          }      }      exit();  }  For part b, we simply adjust it by adding this to affect the priority  int x = 51+i%3;  assigntickets(x); |
| User/psa.c | This file is how we get our data for the graphs  Part A  #include "../include/types.h"  #include "../include/stat.h"  #include "../include/pstat.h"  #include "../user/user.h"  int main() {      struct pstat pTableReal;      struct pstat \*pTable = &pTableReal;      getpinfo(pTable);      for(int i = 0; i < sizeof(pTable->pid); i++){          if(pTable->inuse[i] == 1) {              printf(1, "Use: %d  Tickets: %d  PID: %d  Ticks: %d\n",pTable->inuse[i],pTable->tickets[i],pTable->pid[i],pTable->ticks[i]);          }      }      exit();  }  Part B  #include "../include/types.h"  #include "../include/stat.h"  #include "../include/pstat.h"  #include "../user/user.h"  int main() {      struct pstat pTableReal;      struct pstat \*pTable = &pTableReal;      getpinfo(pTable);      for(int i = 0; i < sizeof(pTable->pid); i++){          if(pTable->inuse[i] == 1) {              printf(1, "Use: %d  Priority: %d  PID: %d  Ticks: %d\n",pTable->inuse[i],pTable->tickets[i],pTable->pid[i],pTable->ticks[i]);          }      }      exit();  } |
| User/user.h | Here we need to add the systems calls and the struct for pstat for both parts  struct pstat;  int getpinfo(struct pstat\*);  int assigntickets(int num); |
| User/usys.S | Add the new syscalls to this file as well.  SYSCALL(getpinfo)  SYSCALL(assigntickets) |
| Lottery Graph |  |
| Priority Graph |  |
| Lessons Learned | In this project, we learned about how to write schedulers for priority and lottery. We learned how systems ticks are more prone to occur when the system is a lottery system rather than in a priority due to the number of interrupts that occur in the scheduler. We have also learned xv6 does not like including certain files in other files such as proc, hence why we have the function for rand inside proc rather than include |
| Elements to Improve for this Project | The big issue with this project is that it probably would not be this difficult if time was actually taken throughout the course with experimenting and coding in the XV6 file alongside the class as part of the course as many have found difficulty with coding in this program, our group included, due to a lack of experience with this program itself and not simply just a lack of understanding c and c++ as we did discover some quirks with XV6 that would not normally be an issue in any C programming language. We believe that by providing more explanation and experience in class on this program when modifying and programming with it could make the overall experience of this project more enjoyable and less of a headache to deal with. |