**LE/EECS 3221 – Operating System Fundamentals**

**Winter 2019**

**Note: For each question/part, paste your code in text and/or the screenshot of the steps, as per requirement.**

|  |  |
| --- | --- |
| **Programming Assignment # 2** | **Submitted By: 212654513** |

|  |  |  |
| --- | --- | --- |
| **Question #** | | **Answer** |
| **#1-A** | | The code:  #include <fcntl.h>  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/types.h>  #include <sys/wait.h>  #include <sys/stat.h>  #include <termios.h>  #include <unistd.h>  #include <signal.h>  int pip[2];  char output[100000];  char inbuf[1000];  /\*  method to handle signal from child process to parent process  prompts to check if the user wishes to diplay the output  \*/  void handle\_signal(int signum){  char answer[1];  printf("\nOutput for \"%s\" is ready. Display it now [Y/N]>>", inbuf);  scanf("%s", answer);  if((strncmp(answer, "Y", 1)) == 0) {  close(pip[1]);  int nbytes = read(pip[0], output, sizeof(output));  printf("%s\n", output);  }  else if(((strncmp(answer, "N", 1)) != 0) && ((strncmp(answer, "Y", 1)) != 0)) {  printf("%s", "Invalid input!");  printf("\nOutput for \"%s\" is ready. Display it now [Y/N]>>", inbuf);  scanf("%s", answer);  if((strncmp(answer, "Y", 1)) == 0) {  close(pip[1]);  int nbytes = read(pip[0], output, sizeof(output));  printf("%s\n", output);  }  }  fseek(stdin,0,SEEK\_END);  printf("SHELL--asma93>>");  fflush(stdout);  }  int main(){  pid\_t pid;  char\* prompt1;  pid\_t ppid;  char input[1000];  char answer[1];  while(((strncmp(input, "exit\n", 1)) != 0))  {  fseek(stdin,0,SEEK\_END);  printf("%s", "SHELL--asma93>>");  fgets(input, 1000, stdin);  prompt1= input;  /\*  Slitting the input string at space  \*/  char \*command1=(char\*)strtok(prompt1, "\n");  strcpy(inbuf, command1);  char \*type = strtok(command1," ");  char \*command = strtok(NULL, "\n");  if((strncmp(type, "BG", 1)) == 0){  signal(SIGUSR1, handle\_signal);  }    char \*array[sizeof(command)];  char \*p = strtok(command," ");  int i = 0;  while (p != NULL) {  array[i++] = p;  p = strtok(NULL, " ");  }  if (pipe(pip)==-1){  printf("%s\n", "Creating pipe failed");  }  if ((pid = fork()) == -1) {  printf("%s\n", "Creating process failed");  }  if(pid == 0) {  //Writing to pipe and directing to the parent using signal handler  dup2 (pip[1], STDOUT\_FILENO);  close(pip[0]);  close(pip[1]);  ppid = getppid();  if((strncmp(type, "BG", 1)) == 0){  kill(ppid, SIGUSR1);  }  execvp(command, array);  /\*  if the first substring is FG then continue in the else block  otherwise continue the parent until signal is recieved  then the signal handler take over  \*/  } else {  if((strncmp(type, "FG", 1)) == 0){  printf("%s", "Output is ready. Display it now [Y/N]>>");  scanf("%s", answer);  if((strncmp(answer, "Y", 1)) == 0) {  close(pip[1]);  int nbytes = read(pip[0], output, sizeof(output));  printf("%s\n", output);  wait(NULL);  }  else if(((strncmp(answer, "N", 1)) != 0) && ((strncmp(answer, "Y", 1)) != 0)) {  printf("%s\n", "Invalid input");  }  }  }  }  return 0;  } |
| **#1-B** | **Screenshots:**  Beginning of the application. Insert “BG ls -lrt”. The parent process will continue and ask for command. Then once the child process execution is complete, the parent will prompt user if the out put should be displayed.    Output of “BG ls -lrt” displayed when user inouts “Y”.  At the end of the display of output, the and then return the prompt to the user.    Nothing displayed if user inputs “N”. The applications moves on and asks for shell command.  If the user inputs anything other than “Y/N” the user is informed about the invalid input and given another chance to input “Y/N”.    If the the user inputs command with “FG” then the prompt is not returned to the user until the execution of the command is complete. | | |
| **#2-A** | **The code:**  #include <stdio.h>  #include <stdlib.h>  #include <limits.h>  #include <pthread.h>  //Queues to store the process for the scheduler  struct Queue\* new\_queue;  struct Queue\* ready\_queue;  //Process with id and time  struct Process  {  int id, time;  };  struct Queue  {  int head, tail, size;  unsigned max;  struct Process\* array;  };  //method to create a queue  struct Queue\* create\_queue(unsigned max)  {  struct Queue\* queue = (struct Queue\*) malloc(sizeof(struct Queue));  queue->max = max;  queue->head = 0;  queue->tail = max - 1;  queue->size = 0;  queue->array = (struct Process\*) malloc(queue->max \* sizeof(1000000));  return queue;  }  //get the first in element of the queue  struct Process head(struct Queue\* queue)  {  return queue->array[queue->head];  }  //get the last in element of the queue  struct Process tail(struct Queue\* queue)  {  return queue->array[queue->tail];  }  //add element to the end of the queue, as the new tail  void enqueue(struct Queue\* queue, struct Process process)  {  queue->tail = (queue->tail + 1)%queue->max;  queue->array[queue->tail] = process;  queue->size = queue->size + 1;  }  //remove the first added element in the queue  void dequeue(struct Queue\* queue)  {  queue->head = (queue->head + 1)%queue->max;  queue->size = queue->size - 1;  }  //check if queue is empty  int isEmpty(struct Queue\* queue)  {  return (queue->size == 0);  }  //check if queue is full  int isFull(struct Queue\* queue)  {  return (queue->size == queue->max);  }  //the thread handling the long time scheduling  void \*long\_term\_scheduler(){  printf("Starting long term scheduling.\n");  /\*  if the ready queue is not full and new queue is not empty  then add procees to ready queue and remove it from the new queue  then give control to the short term scheduler  \*/  while(isFull(ready\_queue) == 0 && isEmpty(new\_queue) == 0) {  struct Process ready\_process = head(new\_queue);  printf("Moving process from new queue to ready queue. Process ID: %d and Process time: %d\n", ready\_process.id, ready\_process.time);    enqueue(ready\_queue, ready\_process);  dequeue(new\_queue);  }  if(isEmpty(new\_queue) == 1){  printf("New queue is empty. No more process to be transferred to the ready queue.\n");  }  else if(isFull(ready\_queue) == 1){  printf("The ready queue is full. Can't move any new process.\n");  }  printf("Long term scheduling is complete.\n");  pthread\_exit(0);  }  void \*short\_term\_scheduler(){  printf("Starting short term scheduling.\n");  /\*  for 5 times reduce the time of process by 2 if possible else reduce by 1  then give control back to log term scheduler  repeat the previous steps after getting control again  until the time of every process is zero  \*/  for(int i = 0; i < 5 && isEmpty(ready\_queue) == 0; i++) {  struct Process ready\_process = head(ready\_queue);  dequeue(ready\_queue);  int process\_time = ready\_process.time;  if(process\_time - 2 > 0){  printf("Reducing time of process by 2 and adding it back to the queue. Process ID: %d and Process time: %d\n", ready\_process.id, ready\_process.time - 2);  ready\_process.time = process\_time - 2;  enqueue(ready\_queue, ready\_process);  }  else if(process\_time - 2 < 0) {  printf("Removing process from ready queue, it's time has reached zero. Process ID: %d and Process time: %d\n", ready\_process.id, ready\_process.time - 1);  ready\_process.time = process\_time - 1;  }  else if(process\_time - 2 == 0) {  printf("Removing process from ready queue, it's time has reached zero. Process ID: %d and Process time: %d\n", ready\_process.id, ready\_process.time - 2);  }  }  printf("Short term scheduling is complete.\n");  pthread\_exit(0);  }  int main()  {  int time;  struct Process process;  pthread\_t scheduler1, scheduler2;  new\_queue = create\_queue(100);  ready\_queue = create\_queue(5);  //populate the new queue wwith 100 processes  for(int id = 1; id < 101; id++){  time = rand() % 31;  process.id = id;  process.time = time;  enqueue(new\_queue, process);  }  while(isEmpty(new\_queue) == 0 || isEmpty(ready\_queue) == 0)  {  //create threads  pthread\_create(&scheduler1, NULL, long\_term\_scheduler, (void \*)&scheduler1);  pthread\_join(scheduler1, NULL);  printf("\n");  pthread\_create(&scheduler2, NULL, short\_term\_scheduler, (void \*)&scheduler2);  pthread\_join(scheduler2, NULL);  printf("\n");  }  printf("Program ended. All processes have zero time left.\n");  return 0;  } | | |
| **#2-B** | **Screenshots:**  I have includes the screen shots of beginning and end of the output on the execution. | | |