Department of Computer Science & Engineering

|  |  |  |  |
| --- | --- | --- | --- |
| **Subject Code:** | **CSL67** | **TERM: Feb-June 2024** | |
| **Subject Name:** | **Unix System Programming & Compiler Design Laboratory** | **Faculty In-charge:** | **CP/PN/SB** |
| **Credits:** | **0:0:1** | **Semester :** | **VI** |

|  |  |
| --- | --- |
| **Exercise Number** | **Problem Statements** |
| 1. | 1. Write a C program to display the file content in reverse order using lseek system call.   #include <fcntl.h>  #include <unistd.h>  #include <stdio.h>  #include <stdlib.h>  int main(int argc,char \*argv[])  {  char ch[1];  if(argc<2)  {  printf("Specify file name !\n");  exit(0);  }  int fin=open(argv[1],O\_RDONLY);  int beg=lseek(fin,0,SEEK\_SET);  int c=0;  int end=lseek(fin,0,SEEK\_END);  while(end>=beg)  {  c++;  read(fin,ch,1);  printf("%s",ch);  end=lseek(fin,-1\*c,SEEK\_END);  }  }   1. Write a C program to create a child process and show how parent and child processes will share the text file and justify that both parent and child shares the same file offset.   #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <fcntl.h>  #include <sys/types.h>  #include <sys/wait.h>  int main(int argc, char \*argv[]) {  int fd;  pid\_t pid;  char buffer[100];  fd = open(argv[1], O\_RDONLY);  if (fd < 0) {  perror("Failed to open file");  exit(EXIT\_FAILURE);  }  pid = fork();  if (pid < 0) {  perror("fork failed");  exit(EXIT\_FAILURE);  } else if (pid == 0) {  ssize\_t bytesRead = read(fd, buffer,5);  if (bytesRead < 0) {  perror("Child failed to read from file");  exit(EXIT\_FAILURE);  }  buffer[bytesRead] = '\0';  printf("Child read: %s\n", buffer);  } else {  wait(NULL);  ssize\_t bytesRead = read(fd, buffer,10);  if (bytesRead < 0) {  perror("Parent failed to read from file");  exit(EXIT\_FAILURE);  }  buffer[bytesRead] = '\0';  printf("Parent read: %s\n", buffer);  close(fd);  }  return 0;  } |
| 2. | 1. Write a C program to display various details of a file using stat structure (Atleast 5 fields).   #include <unistd.h>  #include <stdio.h>  #include <sys/stat.h>  #include <sys/types.h>  int main(int argc, char \*\*argv)  {  if (argc != 2)  return 1;  struct stat fileStat;  if (stat(argv[1], &fileStat) < 0)  return 1;  printf("Information for %s\n", argv[1]);  printf("---------------------------\n");  printf("File Size: \t\t%d bytes\n", fileStat.st\_size);  printf("Number of Links: \t%d\n", fileStat.st\_nlink);  printf("File inode: \t\t%d\n", fileStat.st\_ino);  printf("File Permissions: \t");  printf((S\_ISDIR(fileStat.st\_mode)) ? "d" : "-");  printf((fileStat.st\_mode & S\_IRUSR) ? "r" : "-");  printf((fileStat.st\_mode & S\_IWUSR) ? "w" : "-");  printf((fileStat.st\_mode & S\_IXUSR) ? "x" : "-");  printf((fileStat.st\_mode & S\_IRGRP) ? "r" : "-");  printf((fileStat.st\_mode & S\_IWGRP) ? "w" : "-");  printf((fileStat.st\_mode & S\_IXGRP) ? "x" : "-");  printf((fileStat.st\_mode & S\_IROTH) ? "r" : "-");  printf((fileStat.st\_mode & S\_IWOTH) ? "w" : "-");  printf((fileStat.st\_mode & S\_IXOTH) ? "x" : "-");  printf("\n\n");  printf("The file %s a symbolic link\n", (S\_ISLNK(fileStat.st\_mode)) ? "is" : "is not");  return 0;  }   1. Write a C program that takes the file descriptor as an argument and prints the description of selected file flags for that descriptor.   #include <stdio.h>  #include <stdlib.h>  #include <fcntl.h>  #include <unistd.h>  void print\_file\_flags(int fd);  int main(int argc, char \*argv[]) {  if (argc != 2) {  fprintf(stderr, "Usage: %s <file\_descriptor>\n", argv[0]);  exit(EXIT\_FAILURE);  }  int fd = atoi(argv[1]);  int flags = fcntl(fd, F\_GETFL);  if (flags == -1) {  perror("fcntl");  exit(EXIT\_FAILURE);  }  printf("File descriptor: %d\n", fd);  printf("File flags:\n");  switch (flags & O\_ACCMODE) {  case O\_RDONLY:  printf(" Read only\n");  break;  case O\_WRONLY:  printf(" Write only\n");  break;  case O\_RDWR:  printf(" Read/Write\n");  break;  default:  printf(" Unknown access mode\n");  break;  }  if (flags & O\_APPEND) printf(" Append\n");  if (flags & O\_NONBLOCK) printf(" Non-blocking\n");  if (flags & O\_SYNC) printf(" Synchronous writes\n");  if (flags & O\_DSYNC) printf(" Synchronous data writes\n");  if (flags & O\_RSYNC) printf(" Synchronous reads\n");  if (flags & O\_CREAT) printf(" Create\n");  if (flags & O\_TRUNC) printf(" Truncate\n");  if (flags & O\_EXCL) printf(" Exclusive\n");  if (flags & O\_NOCTTY) printf(" No terminal control\n");  if (flags & O\_CLOEXEC) printf(" Close-on-exec\n");  return 0;  } |
| 3. | 1. Write a C program to simulate system function.   #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/wait.h>  int main() {  const char \*command = "ls";  if (command == NULL) {  fprintf(stderr, "Command is NULL\n");  return 1;  }  pid\_t pid = fork();  if (pid == -1) {  perror("fork failed");  return -1;  } else if (pid == 0) {  // Child process  execl("/bin/sh", "sh", "-c", command, (char \*)NULL);  // If execl() fails  \_exit(EXIT\_FAILURE);  } else {  // Parent process  int status;  if (waitpid(pid, &status, 0) == -1) {  perror("waitpid failed");  return -1;  } else {  printf("Command executed with exit status: %d\n", WEXITSTATUS(status));  return status;  }  }  }   1. Write a C program to implement ls –li command which list the files in a specified directory. Your program should Print 5 attributes of files.   #include <stdio.h>  #include <unistd.h>  #include <fcntl.h>  #include <dirent.h>  #include <time.h>  #include <sys/stat.h>  int main(int argc, char \*argv[])  {  struct dirent \*dir;  struct stat mystat;  DIR \*dp;  dp = opendir(".");  if (dp)  {  while ((dir = readdir(dp)))  {  stat(dir->d\_name, &mystat);  // inode mode uid guid access\_time  printf("%llu %o %d %d %s %s\n", mystat.st\_ino, mystat.st\_mode, mystat.st\_uid, mystat.st\_gid, ctime(&mystat.st\_atime), dir->d\_name);  }  }  } |
| 4. | 1. Write a C program to demonstrate the creation of soft links and the various properties of hard links.   #include <stdio.h>  #include <fcntl.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/stat.h>  int main(int argc, char \*argv[])  {  if (argc == 3)  {  printf("Hard linking %s and %s", argv[1], argv[2]);  if (link(argv[1], argv[2]) == 0)  printf("\nHard link created");  else  printf("\nLink not created");  }  else if (argc == 4)  {  printf("Soft linking %s and %s", argv[1], argv[2]);  if (symlink(argv[1], argv[2]) == 0)  printf("\nSoft link created");  else  printf("\nLink not created");  }  }   1. Write a C program to    1. To create a child process.    2. Child should execute an interpreter file by passing few arguments and some environment variables.    3. Parent should execute an interpreter file by passing few arguments    4. Create an interpreter file that has the path of echoall.c file    5. Create echoall.c file which prints the arguments and environment variables received through parent and child process |
| 5. | 1. Write a program to copy access and modification time of a file to another file using utime function.   #include <stdio.h>  #include <sys/stat.h>  #include <sys/types.h>  #include <unistd.h>  #include <utime.h>  #include <time.h>  #include <fcntl.h>  int main(int argc, char \*argv[]) //copying ctime and mtime of argv[2] to argv[1]  {  int fd;  struct stat statbuf\_1;  struct stat statbuf\_2;  struct utimbuf times;  if (stat(argv[1], &statbuf\_1) < 0)  printf("Error!\n");  if (stat(argv[2], &statbuf\_2) < 0)  printf("Error!\n");  printf("Before Copying ...\n");  printf("Access Time %s\nModification Time%s\n", ctime(&statbuf\_1.st\_atime),  ctime(&statbuf\_1.st\_mtime));  times.modtime = statbuf\_2.st\_mtime;  times.actime = statbuf\_2.st\_mtime;  if (utime(argv[1], &times) < 0)  printf("Error copying time \n");  if (stat(argv[1], &statbuf\_1) < 0)  printf("Error!\n");  printf("After Copying ...\n");  printf("Access Time %s\nModification Time%s\n", ctime(&statbuf\_1.st\_atime), ctime(&statbuf\_1.st\_mtime));  }   1. Write a C program using sigaction system call which calls a signal handler on SIGINT signal and then reset the default action of the SIGINT signal.   #include <stdio.h>  #include <unistd.h>  #include <signal.h>  struct sigaction sig;  void handler(int val)  {  printf("Interrupt Received!\n");  sig.sa\_handler = SIG\_DFL;  sigaction(SIGINT,&sig,0);  }  int main()  {  sig.sa\_flags = 0;  sigemptyset(&sig.sa\_mask);  sigaddset(&sig.sa\_mask,SIGINT); // listen only for SIGNIT  sig.sa\_handler = handler;  sigaction(SIGINT,&sig,0);  while(1)  {  printf("Progress is Happiness!\n");  sleep(1);  }  } |

|  |  |
| --- | --- |
| 6. | 1. Write a C program to remove empty files from the given directory.   #include <stdio.h>  #include <fcntl.h>  #include <unistd.h>  #include <dirent.h>  int main()  {  DIR \*dp;  struct dirent \*dir;  int fd, n;  dp = opendir("."); //open current directory  if (dp)  {  while ((dir = readdir(dp)) != NULL)  {  fd = open(dir->d\_name, O\_RDWR, 0777);  n = lseek(fd, 0, SEEK\_END);  if (!n)  unlink(dir->d\_name);  }  }  }   1. Consider the last 100 bytes as a region. Write a C program to check whether the region is locked or not. If the region is locked, print pid of the process which has locked. If the region is not locked, lock the region with an exclusive lock, read the last 50 bytes and unlock the region.   #include<unistd.h>  #include<stdio.h>  #include<fcntl.h>  #include<sys/types.h>  #include<string.h>  int main(int argc,char \* argv[])  {  struct flock fc;  int fd;  char buf[256];  fd=open(argv[1],O\_RDWR);  if(fd==-1)  {  printf("Error\n");  return 1;  }  lseek(fd,-100,SEEK\_CUR);  fc.l\_whence=SEEK\_CUR;  fc.l\_start=0;  fc.l\_len=100;  if(fcntl(fd,F\_GETLK,&fc)==0)  {  printf("File locked\n");  printf("PID of the process = %d\n",fc.l\_pid);  return 0;  }  else{  printf("Not locked\n");  fc.l\_type=F\_WRLCK;  fc.l\_whence=SEEK\_END;  fc.l\_start=0;  fc.l\_len=100;  if(fcntl(fd,F\_SETLK,&fc)==-1)  {  printf("Error in locking file\n");  return 1;  }  else{  printf("File is locked exclusively\n");  printf("PID of the process = %d\n",fc.l\_pid);  lseek(fd,-50,SEEK\_END);  printf("Contents of the locked file : \n");  buf[50]='\0';  if(read(fd,buf,sizeof(buf)))  printf("%s",buf);  }  }  fc.l\_type=F\_UNLCK;  fc.l\_whence=SEEK\_END;  fc.l\_start=0;  fc.l\_len=100;  if(fcntl(fd,F\_SETLKW,&fc)==-1)  {  printf("Error\n");  return 1;  }  else{  printf("\nFile unlocked\n");  return 0;  }  }  /\*  Output :  touch a  chmod 777 a  cc prog3.c -o prog3  ./prog3  Not locked  File is locked exclusively  PID of the process = 0  Contents of the locked file :  11111111110000000000111111111100000000001111111111  File unlocked  \*/  /\*  Contents of a :  00000000001111111111000000000011111111110000000000  11111111110000000000111111111100000000001111111111  \*/ |
| 7. | 1. Write a C program to illustrate the effect of setjmp and longjmp functions on register and volatile variables.   #include <setjmp.h>  #include <stdio.h>  #include <stdlib.h>  static void f1(int, int, int, int);  static void f2(void);  static jmp\_buf jmpbuffer;  static int globval;  int main(void)  {  int autoval;  register int regival;  volatile int volaval;  static int statval;  globval = 1;  autoval = 2;  regival = 3;  volaval = 4;  statval = 5;  if (setjmp(jmpbuffer) != 0)  {  printf("after longjmp:\n");  printf("globval = %d, autoval = %d, regival = %d, volaval = %d, statval = %d\n", globval, autoval, regival, volaval, statval);  exit(0);  }  /\*  Change variables after setjmp, but before longjmp.  \*/  globval = 95;  autoval = 96;  regival = 97;  volaval = 98;  statval = 99;  f1(autoval, regival, volaval, statval); /\* never returns \*/  exit(0);  }  static void f1(int i, int j, int k, int l)  {  printf("in f1():\n");  printf("globval = %d, autoval = %d, regival = %d, volaval = %d, statval = %d\n", globval, i, j, k, l);  globval = 10000;  j = 10000;  f2();  }  static void f2(void)  {  longjmp(jmpbuffer, 1);  }   1. C program to simulate copy command by accepting the filenames from command line. Report all errors.   #include <stdio.h>  #include <fcntl.h>  #include <unistd.h>  #include <stdlib.h>  int main(int argc, char \*argv[])  {  char buf[100];  int fd1, fd2;  off\_t size, ret, set;  ssize\_t readdata, writedata;  if (argc < 3)  printf("TOO FEW ARGUMENTS");  fd1 = open(argv[1], O\_RDONLY); //Open file 1  if (fd1 == -1)  printf("ERROR IN OPENING FILE: FILE DOES NOT EXIST \n");  else  printf("FILE 1 OPENED SUCCESSFULLY \n");  fd2 = open(argv[2], O\_WRONLY | O\_CREAT | O\_TRUNC, 0666);  if (fd2 == -1)  printf("ERROR IN OPENING FILE");  else  printf("FILE 2 OPENED SUCCESSFULLY \n");  size = lseek(fd1, 0L, SEEK\_END); //obtain the size of file 1 using lseek  if (size == -1)  printf("ERROR: COULD NOT OBTAIN FILE SIZE \n");  else  printf("FILE SIZE OF FILE 1 OBTAINED \n");  ret = lseek(fd1, 0L, SEEK\_SET); //change the current pointer to the beginning of the file  if (ret == -1)  printf("RETRACE FAILED \n");  readdata = read(fd1, buf, size); //read data equal to the size of the first file  if (readdata == -1)  printf("ERROR IN READING FILE CONTENTS \n");  writedata = write(fd2, buf, size); //write the data to file 2 from buffer after read  if (writedata != size)  printf("ERROR IN COPYING FILE");  else  printf("FILE COPIED SUCCESSFULLY");  return 0;  } |
| 8. | 1. Write a C program to remove empty files from the given directory.   #include <stdio.h>  #include <fcntl.h>  #include <unistd.h>  #include <dirent.h>  int main()  {  DIR \*dp;  struct dirent \*dir;  int fd, n;  dp = opendir("."); //open current directory  if (dp)  {  while ((dir = readdir(dp)) != NULL)  {  fd = open(dir->d\_name, O\_RDWR, 0777);  n = lseek(fd, 0, SEEK\_END);  if (!n)  unlink(dir->d\_name);  }  }  }   1. Write a C program to perform the following operations    1. To create a child process    2. The child process should execute a program to show the use of the access function    3. Parent process should wait for the child process to exit    4. Also print the necessary process IDs   #include <stdio.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/wait.h>  #include <stdlib.h>  int main() {  pid\_t pid;  int status;  // Step 1: Fork a child process  pid = fork();  if (pid < 0) {  perror("Fork failed");  return 1;  } else if (pid == 0) {  // Child process  printf("Child process ID: %d\n", getpid());  // Example use of access function  char \*file\_path = "/etc/passwd";  if (access(file\_path, R\_OK) == 0) {  printf("Child process: File %s is readable.\n", file\_path);  } else {  printf("Child process: File %s is not readable.\n", file\_path);  }  exit(0);  } else {  // Parent process  printf("Parent process ID: %d\n", getpid());  printf("Parent process is waiting for child process to exit...\n");  // Step 2: Wait for the child process to exit  waitpid(pid, &status, 0);  printf("Child process exited with status %d.\n", status);  printf("Parent process continues.\n");  }  return 0;  } |
| 9. | 1. Write a C programs to demonstrate usage of umask and chmod functions.   #include <stdio.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <unistd.h>  #include <stdlib.h>  void print\_permissions(const char \*file) {  struct stat st;  if (stat(file, &st) == -1) { perror("stat"); exit(EXIT\_FAILURE); }  printf("Permissions for %s: %o\n", file, st.st\_mode & 0777);  }  int main() {  const char \*filename = "example.txt";  umask(0022); // Set umask  // Create file with initial permissions  int fd = open(filename, O\_CREAT | O\_WRONLY, 0666);  if (fd == -1) { perror("open"); exit(EXIT\_FAILURE); }  close(fd);  print\_permissions(filename); // Print initial permissions  // Change file permissions  if (chmod(filename, 0644) == -1) { perror("chmod"); exit(EXIT\_FAILURE); }  print\_permissions(filename); // Print updated permissions  // Clean up  if (unlink(filename) == -1) { perror("unlink"); exit(EXIT\_FAILURE); }  return 0;  }   1. Write a C program    1. To read first 20 characters from a file    2. seek to 10th byte from the beginning and display 20 characters from there    3. seek 10 bytes ahead from the current file offset and display 20 characters    4. Display the file size   #include<stdlib.h>  #include<unistd.h>  #include<fcntl.h>  #include<sys/stat.h>  int main(int argc,char \*argv[]){  if(argc<2){  printf("specify file name\n");  return 1;  }  int fd;  fd=open(argv[1],O\_RDONLY);  if(fd<0){  printf("read error");  return 1;  }  printf("readof fist 20 bytes");  char buffer[21];  int bytes;  bytes=read(fd,buffer,20);  buffer[bytes]='\0';  printf(buffer);  printf("printing 20 bytes after 10\n");  lseek(fd,10,SEEK\_SET);  bytes=read(fd,buffer,20);  buffer[bytes]='\0';  printf(buffer);  printf("printing 20 bytes ahed 10\n");  lseek(fd,10,SEEK\_CUR);  bytes=read(fd,buffer,20);  buffer[bytes]='\0';  printf(buffer);  int fs=lseek(fd,0,SEEK\_END);  printf("file size is : %d ",fs);  } |
| 10. | 1. Write a C program such that it initializes itself as a Daemon Process.   #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  #include <fcntl.h>  void deamon()  {  pid\_t pid;  pid=fork();  if(pid>0)  {  printf("\nPID of child : %d\n",pid);  exit(0);  }  umask(0);  if(chdir("/")<0)  printf("error");  if(setsid()<0)  printf("error");  printf("Created deamon");  }  int main()  {  deamon();  system("ps -aj");  return 0;  }   1. Demonstrate the working of wait and waitpid system calls with a program   #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/wait.h>  int main() {  pid\_t pid1, pid2;  int status;  // Create the first child process  pid1 = fork();  if (pid1 < 0) {  perror("fork");  exit(EXIT\_FAILURE);  }  if (pid1 == 0) {  // First child process  printf("First child process (PID: %d)\n", getpid());  sleep(2); // Simulate some work with sleep  exit(0); // Exit with status 0  }  // Create the second child process  pid2 = fork();  if (pid2 < 0) {  perror("fork");  exit(EXIT\_FAILURE);  }  if (pid2 == 0) {  // Second child process  printf("Second child process (PID: %d)\n", getpid());  sleep(4); // Simulate some work with sleep  exit(0); // Exit with status 0  }  // Parent process  printf("Parent process waiting for any child to finish...\n");  // Wait for any child process to finish  wait(&status);  if (WIFEXITED(status)) {  printf("A child process terminated with exit status: %d\n", WEXITSTATUS(status));  }  printf("Parent process now waiting for second child process (PID: %d)...\n", pid2);  // Wait specifically for the second child process to finish  waitpid(pid2, &status, 0);  if (WIFEXITED(status)) {  printf("Second child process terminated with exit status: %d\n", WEXITSTATUS(status));  }  printf("Parent process finished.\n");  return 0;  } |

Marks Distribution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Conduction and Result** | **Write-up** | **Execution** | **Viva** | **Change of Program** |
| **Part A** | **4M** | **17M** | **7M** | **-5** |
| **Part B** | **4M** | **18M** | **-5** |