| **copying the contents of one file to another and then removing the original file**  #include <sys/types.h>  #include <sys/stat.h>  #include <stdio.h>  #include <fcntl.h>  #include <unistd.h>  int main(int argc, char \*argv[]) {  int fd1, fd2;  char \*file1, \*file2, buf[2];  file1 = argv[1];  file2 = argv[2];  printf("file1=%s file2=%s\n", file1, file2);  fd1 = open(file1, O\_RDONLY, 0777);  fd2 = open(file2, O\_WRONLY | O\_TRUNC, 0777);  while (read(fd1, buf, 1) > 0) {  write(fd2, buf, 1);  }  remove(file1);  close(fd1);  close(fd2);  return 0;  } | **Display OS version, release number, kernel version:**  echo "Operating System Information:"  uname -srv or uname -a  **Display top 10 processes in descending order:**  echo "Top 10 processes in descending order:"  ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem| head  **Display current logged in user and log name:**  echo "Current logged in user:"  whoami  echo "Log name:"  logname  **Display current shell, home directory, operating system type, current path setting, current working directory:**  echo "Current shell:"  echo $SHELL  echo "Home directory:"  echo $HOME  echo "Operating system type:"  uname -s  echo "Current PATH setting:"  echo $PATH  echo "Current working directory:"  pwd | **Implement solution of Producer consumer problem through Semaphore**  #include<stdio.h>  #include<stdlib.h>  int mutex=1,full=0,empty=3,x=0;  int wait(int s){return (--s);}  int signal(int s){return (++s);}  void producer() { mutex=wait(mutex);full=signal(full);empty=wait(empty);x++;printf("\nProducer produces the item %d",x);mutex=signal(mutex); }  void consumer() { mutex=wait(mutex);full=wait(full);empty=signal(empty);printf("\nConsumer consumes item %d",x);x--;mutex=signal(mutex); }  int main(){  int n;  void producer();  void consumer();  int wait(int); | int signal(int);  printf("\n1.Producer\n2.Consumer\n3.Exit");  while(1){  printf("\nEnter your choice:");  scanf("%d",&n);  switch(n){  case 1:  if((mutex==1)&&(empty!=0)){  producer();  }else{  printf("Buffer is full!!"); }  break;  case 2:  if((mutex==1)&&(full!=0)){  consumer();  }else{  printf("Buffer is empty!!"); }  break;  case 3:  exit(0);  break;  default:  break;  } } return 0; } |
| --- | --- | --- | --- |
| **demonstrate the concept of deadlock avoidance through Banker’s Algorithm.**  #include<stdio.h>  int main() {  int n,m,i,j;  printf("Enter number of processes: ");  scanf("%d",&n);  printf("Enter number of resources: ");  scanf("%d",&m);  int alloc[n][m], max[n][m], avail[m], need[n][m], finish[n], safeSeq[n], work[m];  printf("Enter the allocation matrix:\n");  for(i=0;i<n;i++) {  for(j=0;j<m;j++) {  scanf("%d",&alloc[i][j]);  }  }  printf("Enter the max matrix:\n");  for(i=0;i<n;i++) {  for(j=0;j<m;j++) { | scanf("%d",&max[i][j]);  }  }  printf("Enter the available matrix:\n");  for(i=0;i<m;i++) {  scanf("%d",&avail[i]);  }  for(i=0;i<n;i++) {  for(j=0;j<m;j++) {  need[i][j] = max[i][j] - alloc[i][j];  }  }  for(i=0;i<n;i++) {  finish[i] = 0;  safeSeq[i] = -1;  }  for(i=0;i<m;i++) {  work[i] = avail[i]; | }  int count = 0;  while(count<n) {  int found = 0;  for(i=0;i<n;i++) {  if(finish[i]==0) {  int flag = 1;  for(j=0;j<m;j++) {  if(need[i][j]>work[j]) {  flag = 0;  break;  }  }  if(flag) {  found = 1;  break;  }  }  } | }  if(!found) {  printf("System is not in safe state.\n");  return 0;  }  finish[i] = 1;  safeSeq[count] = i;  count++;  for(j=0;j<m;j++) {  work[j] += alloc[i][j];  }  }  printf("Safe sequence is: ");  for(i=0;i<n;i++) {  printf("%d ",safeSeq[i]);  }  printf("\n");  return 0; |
| **First Fit:**  #include <stdio.h>  int main() {  int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;  printf("Enter no. of blocks:");  scanf("%d",&bno);  printf("\nEnter size of each block:\n");  for(i=0;i<bno;i++) {  scanf("%d",&bsize[i]);  flags[i]=0;  }  printf("\nEnter no. of processes:");  scanf("%d",&pno);  printf("\nEnter size of each process:\n");  for(i=0;i<pno;i++) {  scanf("%d",&psize[i]);  }  for(i=0;i<pno;i++) { | for(j=0;j<bno;j++) {  if(flags[j]==0 && bsize[j]>=psize[i]) {  allocation[i]=j;  flags[j]=1;  break;  }  }  }  printf("\nBlock no.\tsize\t\tprocess no.\t\tsize\n");  for(i=0;i<bno;i++) {  printf("%d\t\t%d\t\t",i+1,bsize[i]);  if(flags[i]==1) {  printf("%d\t\t\t%d\n",allocation[i]+1,psize[allocation[i]]);  } else {  printf("Not allocated\n");  }  }  return 0;  } | **Best Fit::**  #include <stdio.h>  int main() {  int part\_size[20], process[20], flag[20];  int n, p, i, j, temp;  printf("Enter number of partitions: ");  scanf("%d", &n);  printf("Enter size of each partition:\n");  for (i = 0; i < n; i++) scanf("%d", &part\_size[i]);  printf("Enter number of processes: ");  scanf("%d", &p);  printf("Enter size of each process:\n");  for (i = 0; i < p; i++) scanf("%d", &process[i]); | for (i = 0; i < n; i++) flag[i] = 0;  for (i = 0; i < n - 1; i++) {  for (j = i + 1; j < n; j++) {  if (part\_size[i] > part\_size[j]) {  temp = part\_size[i];  part\_size[i] = part\_size[j];  part\_size[j] = temp;  } } }  printf("\nProcess\tPartition\tPartition Size\n");  for (i = 0; i < p; i++) {  for (j = 0; j < n; j++) {  if (part\_size[j] >= process[i] && flag[j] == 0) {  flag[j] = 1;  printf("P%d\t%d\t\t%d\n", i, j, part\_size[j]);  break;  } } }  return 0;  } |
| **Worst Fit:**  #include <stdio.h>  int main() {  int part[20], process[20], flag[20];  int n, p, i, j, temp;  printf("Enter number of partitions: ");  scanf("%d", &n);  for (i = 0; i < n; i++) {  printf("Enter size of partition %d: ", i);  scanf("%d", &part[i]);  }  printf("Enter number of processes: ");  scanf("%d", &p);  for (i = 0; i < p; i++) {  printf("Enter size of process %d: ", i);  scanf("%d", &process[i]);  }  for (i = 0; i < n; i++) flag[i] = 0; | for (i = 0; i < n - 1; i++) {  for (j = i + 1; j < n; j++) {  if (part[i] < part[j]) {  temp = part[i];  part[i] = part[j];  part[j] = temp;  } } }  printf("\nProcess\tPartition\tPartition Size\n");  for (i = 0; i < p; i++) {  for (j = 0; j < n; j++) {  if (part[j] >= process[i] && flag[j] == 0) {  flag[j] = 1;  printf("P%d\t%d\t\t%d\n", i, j, part[j]);  break;  } }  if (j == n) printf("P%d\t--\t\t--\n", i);  }  return 0;  } | **Linux mv Command:**  #include<stdio.h>  #include<stdlib.h>  int main() {  char ch, source\_file[25], target\_file[25];  FILE \*source, \*target;  printf("Enter name of file to copy: ");  gets(source\_file);  source = fopen(source\_file, "r");  if (source == NULL) {  printf("Press any key to exit...\n");  exit(EXIT\_FAILURE);  }  printf("Enter name of target file: ");  gets(target\_file);  target = fopen(target\_file, "w");  if (target == NULL) {  fclose(source);  printf("Press any key to exit...\n");  exit(EXIT\_FAILURE);  }  while ((ch = fgetc(source)) != EOF)  fputc(ch, target);  printf("File copied successfully.\n");  fclose(source);  fclose(target);  return 0;  } | **Linux ls Command:**  #include<stdio.h> |||| #include<dirent.h>  #include<errno.h> |||| #include<stdlib.h>  void \_ls(const char \*dir, int op\_a, int op\_l) {  struct dirent \*d;  DIR \*dh = opendir(dir);  if (!dh) {  perror((errno == ENOENT) ? "Directory doesn't exist" : "Unable to read directory"); exit(EXIT\_FAILURE);  }  while ((d = readdir(dh)) != NULL) {  if (!op\_a && d->d\_name[0] == '.') continue;  printf("%s%s", d->d\_name, op\_l ? "\n" : "");  }  if (!op\_l) printf("\n"); closedir(dh); }  int main(int argc, const char \*argv[]) {  if (argc == 1) \_ls(".", 0, 0);  else if (argc == 2 && argv[1][0] == '-') {  int op\_a = 0, op\_l = 0;  for (const char \*p = argv[1] + 1; \*p; p++) {  if (\*p == 'a') op\_a = 1;  else if (\*p == 'l') op\_l = 1;  else {  perror("Option not available");  exit(EXIT\_FAILURE);  } }  \_ls(".", op\_a, op\_l);  }  return 0;  } |

| **Name** | **Syntax** | **Description** |
| --- | --- | --- |
| pwd | pwd | Print the full filename of the current working directory. |
| ls | ls [OPTION]... [FILE]... | List information about the FILEs. Sort entries alphabetically if no options specified. |
| mkdir | mkdir DIRECTORY(ies) | Create the DIRECTORY(ies), if they do not already exist. |
| rmdir | rmdir [OPTION]... DIRECTORY... | Remove the DIRECTORY(ies), if they are empty. |
| whatis | whatis [command name(s)] | Display one-line manual page descriptions. Searches the manual page names and displays descriptions. |
| man | man [command name(s)] | Display the manual page associated with each given program, utility, or function name. |
| cat | cat [OPTION]... [FILE]... | Concatenate FILE(s) or standard input to standard output. |
| cp | cp [OPTION] SOURCE DEST | Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY. |
| rm | rm [OPTION]... FILE... | Remove each specified file. By default, does not remove directories. |
| head | head [OPTION]... [FILE]... | Print the first part of files. Print the first 10 lines of each FILE to standard output. |
| tail | tail [OPTION]... [FILE]... | Print the last part of files. Print the last 10 lines of each FILE to standard output. |
| chmod | chmod [OPTION]... MODE[,MODE]... FILE... | Change file mode bits according to mode. |
| cut | cut OPTION... [FILE]... | Print selected parts of lines from each FILE to standard output. |
| grep | grep [OPTIONS] PATTERN [FILE...] | Print lines matching a pattern. Searches the named input FILEs for lines containing a match. |
| wc | wc [OPTION]... [FILE]... | Print newline, word, and byte counts for each FILE. |
| sort | sort [OPTION]... [FILE]... | Write sorted concatenation of all FILE(s) to standard output. |
| cal | cal [month] [year] | Display a calendar. Display the current month if arguments are not specified. |
| cmp | cmp [OPTION]... FILE1 FILE2 | Compare two files byte by byte. |
| date | date [OPTION]... [+FORMAT] | Print or set the system date and time. Display the current time in the given FORMAT. |
| clear | clear | Clear the terminal screen. |