AOS Practical Slips Solutions (Part - I)

Q1.

```
#include<stdio.h>

#include<dirent.h>

#include<string.h>

#include<unistd.h>

int file_exist(char *filename)

{

FILE *fp = fopen(filename, “r”);

if (fp)

{

return 1;

} else {

return 0;

}

}

void main()

{

int  N = file_exist(“abc.txt”);

if (N == 1){

printf(“file does exist in current directory”);

}

else{
```

```
printf(“file does not  exist in current directory”);
}
```

Q2:

```
#include<stdio.h>
#include<signal.h>
#include<stdlib.h>
#include<unistd.h>
void main(){
sigset_t newmask, oldmask, pendmask;
sigemptyset(&newmask);
sigaddset(&newmask, SIGQUIT);
if(sigprocmask(SIG_BLOCK, &newmask, &oldmask) < 0){
printf("SIG_BLOCK error");
exit(1);
}
sleep(5);
if(sigpending(&pendmask) < 0){
printf("sigpending error\n");
exit(1);
}
if(sigismember(&pendmask, SIGQUIT)){
printf("SIGQUIT Pending\n");
}
if(sigprocmask(SIG_SETMASK, &oldmask, NULL) < 0){
printf("SIG_SETMASK error\n");
```

```
exit(1);
}
sleep(5);
exit(0);
}
```

Q3:

```
#include <stdio.h>
#include <dirent.h>
#include<string.h>
#include<unistd.h>
void main(int argc, char *argv[])
{
struct dirent *de;
if(argc != 2){
fprintf(stderr, "usage : %s <search string>\n", argv[0]);
return;
}
DIR *directory = opendir(".");
char **filenames;
if (directory == NULL)
{
printf("Could not open current directory" );
return;
}
char *searchOut;
```

```
rewinddir(directory);
while ((de = readdir(directory)) != NULL)
if(strcmp(de->d_name,".") != 0 && strcmp(de->d_name,"..")){
searchOut = strstr(de->d_name, argv[1]);
int index = searchOut - de->d_name;
if(index == 0){
printf("%s\n", de->d_name);
}
}
closedir(directory);
}
```

Q:4

```
#include<stdio.h>
#include<stdlib.h>
#include <setjmp.h>
static jmp_buf jmpbuffer;
static int globval;
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);
f2();
}
static void f2(void)
```

```
{
longjmp(jmpbuffer, 1);
}
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);
}
```

Q5:

```
#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("%s details:\n",argv[1]);
printf(" file size:%d\n",filestat,st_size);
printf(" No. of hard links:%d\n",filestat,st_nlink);
printf(" file inode:%d\n",filestat,st_ino);
printf(" file permissions :");
printf(S_ISDIR(filestat.st_mode)?"d":"v");
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_IWDTH)?"w":".");
printf( (filestat.st_mode & S_IXOTH)?"x":".");
```

```
printf(“\n”);
char timestr[50];
struct tm *modified_time = localtime(&filestat.st_time);
strftime(timestr, &o, “%b %d %l : %m %p”, modified_time);
printf(“Modified time:%s\n”,timestr);
struct tm *access_time = localtime(&filestat.st_time);
strftime(timestr, &o, “%b %d %l : %m %p”, access_time);
printf(“access time:%s\n”,timestr);
return 0;
}
```

Q6:

```
*/
#include<sys/resource.h>
#include<sys/time.h>
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/wait.h>
void child_process_code(){
int j,k=5000;
char filename[10] = "hello";
FILE *fp = fopen(filename,"a");
for(j=0;j<10;j++){
while(k--);
```

```
k=5000;
fprintf(fp,"%d - ",j);
}
fprintf(fp, "\n");
fclose(fp);
}
void main(){
int status = 0,i;
struct rusage usage;
for(i=0;i<10;i++){
if(fork()==0){
child_process_code();
exit(0);
}
}
while((wait(&status))>0);
getrusage(RUSAGE_CHILDREN, &usage);
printf("Total time spent in user mode by children : %ld s %ld ms\n",
usage.ru_utime.tv_sec, usage.ru_utime.tv_usec);
printf("Total time spent in kernel mode by children : %ld s %ld ms\n",
usage.ru_stime.tv_sec, usage.ru_stime.tv_usec);
}
```

Q7:

```
#include<stdio.h>
#include<stdlib.h>
```

```
#include<sys/stat.h>
#include<sys/types.h>
#include<time.h>
#include<fcntl.h>
int main(int argc, char const *argv[])
{
if(argc != 2){
fprintf(stderr, "usage : %s <filepath>\n", argv[0]);
return 1;
}
int file = open(argv[1], O_RDONLY);
if(file < 0){
fprintf(stderr, "error opening file\n");
return 1;
}
struct stat st;
if(fstat(file, &st) < 0){
fprintf(stderr, "error reading file info\n");
return 1;
}
printf("%s Details : \n", argv[1]);
printf("File size : %ld\n", st.st_size);
printf("Number of hard links : %ld\n", st.st_nlink);
printf("File inode : %ld\n", st.st_ino);
printf("File Permissions : ");
```

```
printf(S_ISDIR(st.st_mode) ? "d" : "-");
printf((st.st_mode & S_IRUSR) ? "r" : "-");
printf((st.st_mode & S_IWUSR) ? "w" : "-");
printf((st.st_mode & S_IXUSR) ? "x" : "-");
printf((st.st_mode & S_IRGRP) ? "r" : "-");
printf((st.st_mode & S_IWGRP) ? "w" : "-");
printf((st.st_mode & S_IXGRP) ? "x" : "-");
printf((st.st_mode & S_IROTH) ? "r" : "-");
printf((st.st_mode & S_IWOTH) ? "w" : "-");
printf((st.st_mode & S_IXOTH) ? "x" : "-");
printf("\n");
char timestr[50];
struct tm *modified_time = localtime(&st.st_mtime);
strftime(timestr, 80, "%b %d %l:%M %p", modified_time);
printf("Modified time : %s\n", timestr);
struct tm *access_time = localtime(&st.st_atime);
strftime(timestr, 80, "%b %d %l:%M %p", access_time);
printf("Access time : %s\n", timestr);
return 0;
}
```

Q8:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
```

```
#include<sys/wait.h>
#include<errno.h>
#include<signal.h>
void sigint(){
}
void main(){
int filedes[2];
if (pipe(filedes) == -1) {
perror("pipe");
exit(1);
}
signal(SIGINT, sigint);
if(fork() == 0){
while ((dup2(filedes[1], STDOUT_FILENO) == -1)) {}
char *args[] = {"ls","-l", NULL};
int ret = execvp("ls",args);
if(ret <0){
printf("Program can't be executed\n");
}
exit(0);
}
close(filedes[1]);
if(fork() == 0){
while((dup2(filedes[0], STDIN_FILENO) == -1)){}
char *args[] = {"wc","-l", NULL};
```

```
int ret = execvp("wc",args);
if(ret <0){
printf("Program can't be executed\n");
}
exit(0);
}
char output[100];
read(filedes[0], output, 100);
printf("%s", output);
close(filedes[0]);
// exit(0);
int i=0;
while(i<2000000000) i++;
}
```

Q9:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<sys/wait.h>
#define MESSAGE_BLOCK_SIZE 20
void main(){
int fd[2];
char message1[MESSAGE_BLOCK_SIZE] = "Hello World";
char message2[MESSAGE_BLOCK_SIZE] = "Hello SPPU";
char message3[MESSAGE_BLOCK_SIZE] = "Linux is Funny";
```

```
char message[MESSAGE_BLOCK_SIZE];
if(pipe(fd) < 0){
printf("Error creating pipe\n");
exit(1);
}
int pid = fork();
if(pid == 0){
write(fd[1], message1, MESSAGE_BLOCK_SIZE);
write(fd[1], message2, MESSAGE_BLOCK_SIZE);
write(fd[1], message3, MESSAGE_BLOCK_SIZE);
exit(0);
}
int status;
while(wait(&status) > 0);
read(fd[0], message, MESSAGE_BLOCK_SIZE);
printf("%s\n", message);
read(fd[0], message, MESSAGE_BLOCK_SIZE);
printf("%s\n", message);
read(fd[0], message, MESSAGE_BLOCK_SIZE);
printf("%s\n", message);
}
```

Q10:

```
#include <stdio.h>
#include <dirent.h>
#include<string.h>
```

```
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
int doesEndWithTxt(char *string){
char *substr = ".txt";
int length = 4;
int lengthLong = strlen(string) - 1;
int doesEnd = 1;
while(length--){
if(string[lengthLong] != substr[length]){
doesEnd = 0;
break;
}
lengthLong--;
}
return doesEnd;
}
void main(int argc, char *argv[])
{
struct dirent *de;
DIR *directory = opendir(".");
char **filenames;
if (directory == NULL)
{
```

```
printf("Could not open current directory" );
return;
}
int mergedfd = open("merged.txt", O_CREAT|O_WRONLY, S_IRUSR |
S_IWUSR);
char block[8];
while ((de = readdir(directory)) != NULL)
if(strcmp(de->d_name,".") != 0 && strcmp(de->d_name,".."){
if(strlen(de->d_name) > 4 && doesEndWithTxt(de->d_name)){
printf("%s\n", de->d_name);
int fd = open(de->d_name, O_RDONLY);
while(read(fd, block, 8) > 0){
write(mergedfd, block, 8);
}
close(fd);
}
}
printf("Merged file : merged.txt\n");
printf("file descriptor : %d\n", mergedfd);
close(mergedfd);
closedir(directory);
}
```

Q11:

```
#include<stdio.h>
#include<stdlib.h>
```

```
#include<fcntl.h>
#include<sys/mman.h>
#include<sys/stat.h>
#define BUFFER_SIZE 100
int main(int argc, char const *argv[])
{
if(argc != 2){
fprintf(stderr, "Please pass file name\n");
return 1;
}
char *addr;
char buffer[BUFFER_SIZE];
int fd = open(argv[1], O_RDONLY);
struct stat st;
if(fstat(fd, &st) < 0){
fprintf(stderr, "Error reading file info\n");
return 1;
}
if((addr = mmap(NULL, st.st_size, PROT_READ, MAP_PRIVATE, fd, 0))
== MAP_FAILED){
fprintf(stderr, "Error mapping file\n");
return 1;
}
for(int i = st.st_size; i >= 0; i--){
printf("%c", addr[i]);
```

```
}
munmap(addr);
close(fd);
return 0;
}
```

Q12:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<sys/wait.h>
#include<dirent.h>
int occurrence(char *str, char ch){
int count = 0;
for (int i = 0; str[i] != '\0'; ++i) {
if (ch == str[i])
++count;
}
return count;
}
void list_process(char **arguments, int argc){
if(argc != 3){
printf("Invalid arguments\n");
exit(1);
}
```

```
DIR *directory = opendir(arguments[2]);
struct dirent *de;
char **filenames;
if (directory == NULL)
{
printf("Could not open current directory" );
return;
}
if(strcmp(arguments[1],"f")==0){
while ((de = readdir(directory)) != NULL)
if(strcmp(de->d_name,".") != 0 && strcmp(de->d_name,"..")){
printf("%s\n", de->d_name);
}
} else if(strcmp(arguments[1],"n")==0){
int fileCount = 0;
while ((de = readdir(directory)) != NULL)
if(strcmp(de->d_name,".") != 0 && strcmp(de->d_name,"..")){
fileCount++;
}
printf("Total files : %d\n", fileCount);
} else if(strcmp(arguments[1],"i")==0){
while ((de = readdir(directory)) != NULL)
if(strcmp(de->d_name,".") != 0 && strcmp(de->d_name,"..")){
printf("%s -> %ld\n",de->d_name, de->d_ino);
}
```

```
}
closedir(directory);
}
void main(){
char *cmd = (char*) malloc(100 * sizeof(char));
char *delimeter = " ";
int status;
char username[20];
getlogin_r(username, 20);
while(1){
printf("%s$ ",username);
fgets(cmd, 100, stdin);
if(cmd[strlen(cmd)-1] == '\n'){
cmd[strlen(cmd)-1] = '\0';
}
if(cmd[strlen(cmd)-1] == ' '){
cmd[strlen(cmd)-1] = '\0';
}
int occur = occurrence(cmd,' ');
int argc = occur +1;
char **arguments = (char**) malloc((argc + 1)*sizeof(char*));
int i=0;
char *token = strtok(cmd, delimeter);
int len = strlen(token);
arguments[i] = malloc(len);
```

```
strcpy(arguments[i],token);
i++;
while(i <= occur){
token = strtok(NULL, delimeter);
int len = strlen(token);
arguments[i] = malloc(len);
strcpy(arguments[i],token);
i++;
}
if(fork()==0){
if(strcmp(arguments[0],"list") == 0){
list_process(arguments, argc);
exit(0);
} else{
execvp(arguments[0],arguments);
}
}
while(wait(&status)>0);
free(arguments);
}
}
```

Q13:

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
```

```
#include<unistd.h>
void main(){
int fd = open("hole.txt",O_CREAT|O_RDWR);
system("chmod 722 hole.txt");
char message[] = "This is a demonstration";
char message2[] = " for hole in a file.";
write(fd, message, sizeof(message)); //write first part
lseek(fd, 10, SEEK_END); //adding hole of 10 characters
write(fd, message2, sizeof(message2));//write second part
system("od -c hole.txt");
}
```

Q14:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<pwd.h>
#include<grp.h>
#include<stdlib.h>
#include<time.h>
#include<string.h>
void print_permissions(struct stat filestat){
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" : "-");
}
void main(int argc, char **argv){
if(argc < 2){
printf("No file name provided");
exit(1);
}
char *filename = argv[1];
struct stat filestat;
int ret = stat(filename, &filestat);
if(ret < 0){
printf("Error getting file info.\n");
}
struct passwd *pw = getpwuid(filestat.st_uid);
struct group *gw = getgrgid(filestat.st_gid);
struct tm *modified_time = localtime(&filestat.st_mtime);
time_t ctime = time(NULL);
struct tm *current_time = localtime(&ctime);
print_permissions(filestat);
```

```
printf(" %ld %s %s %ld ",filestat.st_nlink, pw->pw_name, gw->gr_name,
filestat.st_size);
char timestr[80];
if(modified_time->tm_year == current_time->tm_year){
strftime(timestr, 80, "%b %d %l:%M %p", modified_time);
printf("%s ", timestr);
} else{
strftime(timestr, 80, "%b %d", modified_time);
printf("%s %d ", timestr, modified_time->tm_year);
}
printf("%s\n", filename);
}
```

Q15:

```
#include <stdio.h>
#include <sys/resource.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int main() {
struct rlimit old_lim, lim, new_lim;
// Get old limits
if( getrlimit(RLIMIT_NOFILE, &old_lim) == 0)
```

```
printf("Old limits -> soft limit= %ld \t"
" hard limit= %ld \n", old_lim.rlim_cur,
old_lim.rlim_max);
else
fprintf(stderr, "%s\n", strerror(errno));
// Set new value
lim.rlim_cur = 3;
lim.rlim_max = 1024;
// Set limits
if(setrlimit(RLIMIT_NOFILE, &lim) == -1)
fprintf(stderr, "%s\n", strerror(errno));
// Get new limits
if( getrlimit(RLIMIT_NOFILE, &new_lim) == 0)
printf("New limits -> soft limit= %ld "
"\t hard limit= %ld \n", new_lim.rlim_cur,
new_lim.rlim_max);
else
fprintf(stderr, "%s\n", strerror(errno));
return 0;
}
```

Q16:

```
#include <stdio.h>
#include <dirent.h>
#include<string.h>
#include<unistd.h>
```

```
#include<time.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<stdlib.h>
typedef struct file_info{
char *name;
size_t size;
}fileinfo;
void insertionSort(fileinfo info[], int n)
{
int i, j;
fileinfo key;
for (i = 1; i < n; i++)
{
key = info[i];
j = i - 1;
while (j >= 0 && info[j].size > key.size)
{
info[j + 1] = info[j];
j = j - 1;
}
info[j + 1] = key;
}
}
void main(int argc, char **argv)
```

```
{
struct stat fstat;
if(argc < 2){
printf("no files passed\n");
exit(1);
}
int fileCount = argc -1;
fileinfo info[fileCount];
int i;
for(i =1;i<argc;i++){
info[i-1].name = argv[i];
stat(argv[i],&fstat);
info[i-1].size = fstat.st_size;
}
insertionSort(info, fileCount);
for(i=0;i<fileCount;i++){
printf("%s -> %ld\n", info[i].name, info[i].size);
}
}
```

Q17:

```
#include <stdio.h>
#include <stdlib.h>
int main () {
printf("Old Environment variables\n");
```

```
printf("PATH : %s\n", getenv("PATH"));
printf("HOME : %s\n", getenv("HOME"));
printf("ROOT : %s\n", getenv("ROOT"));
setenv("PATH","",1);
setenv("HOME","",1);
setenv("ROOT","",1);
printf("New Environment variables\n");
printf("PATH : %s\n", getenv("PATH"));
printf("HOME : %s\n", getenv("HOME"));
printf("ROOT : %s\n", getenv("ROOT"));
return(0);
}
```

Q18:

```
#include <stdio.h>
#include <dirent.h>
#include<string.h>
#include<unistd.h>
#include<malloc.h>
void insertionSort(char *arr[], int n)
{
int i, j;
char *key;
for (i = 1; i < n; i++)
{
key = arr[i];
```

```
j = i - 1;
while (j >= 0 && strcmp(arr[j], key) > 0)
{
arr[j + 1] = arr[j];
j = j - 1;
}
arr[j + 1] = key;
}
}
void main()
{
struct dirent *de;
DIR *directory = opendir(".");
char **filenames;
if (directory == NULL)
{
printf("Could not open current directory" );
return;
}
char **subdirectories;
int dirCount = 0;
while ((de = readdir(directory)) != NULL)
if(strcmp(de->d_name,".") && strcmp(de->d_name,"..") && de->d_type
== DT_DIR){
dirCount++;
```

```
}
subdirectories = (char **) malloc(sizeof(char*) * dirCount);
rewinddir(directory);
int i=0;
while ((de = readdir(directory)) != NULL)
if(strcmp(de->d_name,".") && strcmp(de->d_name,"..") && de->d_type
== DT_DIR){
subdirectories[i] = de->d_name;
i++;
}
closedir(directory);
insertionSort(subdirectories, dirCount);
for(i=0;i<dirCount;i++){
printf("%s\n", subdirectories[i]);
}
}
```

Q19:

```
#include<malloc.h>
#include<stdio.h>
void main(){
struct mallinfo minfo;
minfo = mallinfo();
printf("Memory Allocation Statistics \n");
printf("Non-mmapped space allocated (bytes) : %d\n", minfo.arena);
printf("Number of free chunks : %d\n", minfo.ordblks);
```

```
printf("Number of free fastbin blocks : %d\n", minfo.smblks);
printf("Number of mmapped regions : %d\n", minfo.hblks);
printf("Space allocated in mmapped regions (bytes) : %d\n", minfo.hblkhd);
printf("Maximum total allocated space (bytes) : %d\n", minfo.usmblks);
printf("Space in freed fastbin blocks (bytes) : %d\n", minfo.fsmblks);
printf("Total allocated space (bytes) : %d\n", minfo.uordblks);
printf("Total free space (bytes) : %d\n", minfo.fordblks);
printf("Top-most, releasable space (bytes) : %d\n", minfo.keepcost);
}
```

Q20:

```
#include<stdio.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<fcntl.h>
#include<stdlib.h>
#include<unistd.h>
void main(int argc, char **argv){
umask(0000);
int fd1 = creat("first.txt", S_IWUSR | S_IRUSR | S_IRGRP | S_IWGRP |
S_IROTH | S_IWOTH);
int fd2 = creat("second.txt", S_IWUSR | S_IRUSR);
struct stat st1;
struct stat st2;
if(fstat(fd1, &st1) < 0 || fstat(fd2, &st2) < 0){
fprintf(stderr, "Error reading file stat\n");
```

```
exit(1);
}
chmod("first.txt", (st1.st_mode | S_ISGID) & ~S_IXGRP);
chmod("second.txt", st2.st_mode | S_IROTH);
close(fd1);
close(fd2);
}
```