AOS Practical Slips Solution - (Part - II)

Q1.

```
#include<stdio.h>
#include<alloca.h>
void main(){
int i, n;
int *a;
printf("Enter the number of elements : ");
scanf("%d",&n);
a = (int*) alloca(n * sizeof(int));
printf("Enter %d elements...\n", n);
for(i=0; i< n;i++){
printf("%d : ", i);
scanf("%d",&a[i]);
}
printf("You entered...\n");
for(i = 0; i < n; i++){
printf("%d\t", a[i]);
}
printf("\n");
}
```

Q2.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<sys/wait.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 count_process(char **arguments, int argc){
if(argc != 3){
printf("Invalid arguments\n");
exit(1);
}
FILE *fp = fopen(arguments[2],"r");
if(strcmp(arguments[1],"c")==0){
int charactercount = 0;
while(fgetc(fp) != EOF) charactercount++;
printf("Characters : %d\n", charactercount);
} else if(strcmp(arguments[1],"w")==0){
int wordcount = 0;
char c;
while((c = fgetc(fp)) != EOF) {
if(c == ' ')
wordcount++;
}
printf("Words : %d\n", wordcount);
} else if(strcmp(arguments[1],"l")==0){
int linecount = 0;
char c;
while((c = fgetc(fp)) != EOF) {
if(c == '\n')
linecount++;
```

```
}
printf("Lines : %d\n", linecount);
}
fclose(fp);
}
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],"count") == 0){
count_process(arguments, argc);
exit(0);
} else{
execvp(arguments[0],arguments);
}
}
while(wait(&status)>0);
free(arguments);
}
}
```

Q3.

```
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <stdlib.h>
void catch(int);
int main (void) {
pid_t pid;
pid = fork();
if(pid < 0){
fprintf(stderr, "Error in fork execution\n");
return 1;
}
```

```
if (pid != 0){
signal(SIGALRM,catch);
pause();
} else{
kill(getppid(),SIGALRM);
}
return 0;
}
void catch(int signo) {
write(STDOUT_FILENO, "alarm is fired\n", 15);
}
```

Q4.

```
#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>
void main(int argc, char **argv)
{
struct dirent *de;
struct stat fstat;
struct tm *timeinfo;
if(argc != 2){
printf("no size value passed\n");
exit(1);
}
int size = atoi(argv[1]);
```

```
if(size <0){
printf("invalid size value : size should be non negative\n");
exit(1);
}
DIR *directory = opendir(".");
char **filenames;
if (directory == NULL)
{
printf("Could not open current directory" );
return;
}
while ((de = readdir(directory)) != NULL)
if(strcmp(de->d_name,".") != 0 && strcmp(de->d_name,"..")){
stat(de->d_name,&fstat);
if(fstat.st_size > size){
printf("%s\n",de->d_name );
}
}
closedir(directory);
}
```

Q5.

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<unistd.h>
void main(){
int fd;
fd = open("output.txt",O_CREAT| O_WRONLY, 0777);
close(STDOUT_FILENO);
dup(fd);
printf("this is some text to be printed on the screen\n");
```

```
printf("but it will be written to the file output.txt\n");
}
```

Q6.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<sys/wait.h>
#include<dirent.h>
#define max(x,y) ((x) > (y))? (x) : (y);
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 print_first_lines(char *linesString, FILE *fp){
char c;
int lines = atoi(linesString);
int lineCount = 1;
printf("%3d |", lineCount );
while((c = fgetc(fp)) != EOF){
fputc(c, stdout);
if(c == '\n') {
lineCount++;
if(lineCount > lines){
break;
}
printf("%3d |", lineCount );
```

```
}
}
printf("\n");
}
void print_last_lines(char *linesString, FILE *fp){
char c;
int lines = atoi(linesString);
int lineCount = 0;
while((c = fgetc(fp)) != EOF){
if(c == '\n') lineCount++;
}
int startLine = max(0,lineCount + lines);
lineCount = 0;
fseek(fp, 0, SEEK_SET);
while((c = fgetc(fp)) != EOF){
if(lineCount > startLine){
fputc(c, stdout);
}
if(c == '\n'){
lineCount++;
if(lineCount > startLine){
printf("%3d |", lineCount);
}
}
}
printf("\n");
}
void print_all_lines(FILE *fp){
char c;
while((c = fgetc(fp)) != EOF){
fputc(c, stdout);
```

```
}
printf("\n");
}
void typeline_process(char **arguments, int argc){
if(argc != 3){
printf("Invalid arguments\n");
exit(1);
}
FILE *fp = fopen(arguments[2],"r");
if(arguments[1][0] == '+'){
print_first_lines(arguments[1], fp);
} else if(arguments[1][0] == '-'){
print_last_lines(arguments[1], fp);
} else if(strcmp(arguments[1],"a")==0){
print_all_lines(fp);
}
fclose(fp);
}
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],"typeline") == 0){
typeline_process(arguments, argc);
exit(0);
} else{
execvp(arguments[0],arguments);
}
}
while(wait(&status)>0);
free(arguments);
}
}
```

Q7.

```
#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);
}
```

Q8.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<string.h>
#include<sys/wait.h>
#include<dirent.h>
#define max(x,y) ((x) > (y))? (x) : (y);
#define RESET "\033[0m"
#define GREEN "\033[32m"
int occurrence(char *str, char ch){
int count = 0;
for (int i = 0; str[i] != '\0'; ++i) {
if (ch == str[i])
++count;
}
return count;
}
int search(char* pattern, char* text, int lineNo, int first, int count)
{
int M = strlen(pattern);
int N = strlen(text);
int found = 0, i, j;
/* A loop to slide pat[] one by one */
for (i = 0; i <= N - M; i++) {
for (j = 0; j < M; j++)
if (text[i + j] != pattern[j])
break;
if (j == M){
found++;
}
```

```
}
if(found && !count){
int printed = 0;
printf("%3d |", lineNo+1);
for(i = 0; i < N; i++){
if(i < N-M){
for (j = 0; j < M; j++)
if (text[i + j] != pattern[j])
break;
}
if(j == M && i< N-M && !(first && printed)){
printf("%s%s%s",GREEN, pattern, RESET);
printed++;
i+=(M-1);
} else{
printf("%c", text[i]);
}
}
}
return found;
}
int count_lines(FILE *fp){
int c;
int lines = 0;
while((c = fgetc(fp))){
if(c == '\n'){
lines++;
}
if(feof(fp)){
break;
}
```

```
}
fseek(fp, 0, SEEK_SET);
return lines;
}
void print_occurrence(char *pattern, FILE *fp, int first, int count){
int lines = count_lines(fp);
int l, found;
char **fileContent = (char**) malloc(lines * sizeof(char*));
//allocate memory for each of those lines according to its character count
for(l=0;l < lines;l++){
fileContent[l] = (char*) malloc(1000 * sizeof(char));
}
for(l=0;l < lines;l++){
fgets(fileContent[l],1000, fp);
}
int totalCount = 0;
for(l=0;l < lines;l++){
if(found = search(pattern, fileContent[l], l, first, count)){
if(first){
return;
}
totalCount +=found;
}
}
if(count){
printf("Total Occurrences : %d\n", totalCount);
}
for(l=0;l < lines;l++){
free(fileContent[l]);
}
free(fileContent);
```

```
}
void search_process(char **arguments, int argc){
if(argc != 4){
printf("Invalid arguments\n");
return;
}
FILE *fp = fopen(arguments[3],"r");
if(fp == NULL){
printf("Error opening file\n");
return;
}
char *pattern = arguments[2];
if(strcmp(arguments[1],"f")==0){
print_occurrence(pattern, fp, 1, 0);
} else if(strcmp(arguments[1],"c")==0){
print_occurrence(pattern, fp, 0, 1);
} else if(strcmp(arguments[1],"a")==0){
print_occurrence(pattern, fp, 0, 0);
}
fclose(fp);
}
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],"search") == 0){
search_process(arguments, argc);
exit(0);
} else{
execvp(arguments[0],arguments);
}
}
while(wait(&status)>0);
free(arguments);
```

```
}
}
Q9.
#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;
}
```

Q10.

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
volatile int sigid = 0;
// function declaration
void sighup(int sig);
void sigint(int sig);
void sigquit(int sig);
// driver code
```

```
void main()
{
int pid, timeCounter;
pid = fork();
if (pid < 0) {
perror("fork");
exit(1);
}
if (pid == 0) { /* child */
signal(SIGHUP, sighup);
signal(SIGINT, sigint);
signal(SIGQUIT, sigquit);
while(1){/* loop for ever */
if(sigid == SIGHUP){
printf("CHILD: I have received a SIGHUP\n");
sigid = 0;
continue;
} else if(sigid == SIGINT){
printf("CHILD: I have received a SIGINT\n");
sigid = 0;
continue;
} else if(sigid == SIGQUIT){
printf("My Daddy has killed me!!!\n");
sigid = 0;
break;
}
}
exit(0);
}
sleep(1);
fflush(stdout);
```

```
while(1){
printf("\nPARENT: sending SIGHUP to process : %d\n\n", pid);
kill(pid, SIGHUP);
timeCounter+=3;
if(timeCounter > 15){
break;
}
sleep(3); /* pause for 3 secs */
printf("\nPARENT: sending SIGINT to process : %d\n\n", pid);
kill(pid, SIGINT);
timeCounter+=3;
if(timeCounter > 15){
break;
}
sleep(3); /* pause for 3 secs */
}
printf("\nPARENT: sending SIGQUIT to process : %d\n\n", pid);
usleep(10000);
kill(pid, SIGQUIT);
sleep(1);
}
void sighup(int sig)
{
sigid = SIGHUP;
}
void sigint(int sig)
{
sigid = SIGINT;
}
void sigquit(int sig)
{
```

```
sigid = SIGQUIT;
}
```

Q11.

```
#include<stdio.h>
#include<stdlib.h>
#include<signal.h>
#include<unistd.h>
void sigint(){
write(STDOUT_FILENO, "Press Ctrl + C once again to exit", 33);
signal(SIGINT, SIG_DFL);
}
void main(){
signal(SIGINT, sigint);
while(1);
}
```

Q12.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/wait.h>
#include<errno.h>
void main(){
int fd = open("output.txt",O_CREAT|O_RDWR);
if(fork() == 0){
if(dup2(fd, STDOUT_FILENO) == -1){
printf("Invalid file discriptor\n");
}
char *args[] = {"ls","-l", NULL};
int ret = execvp("ls",args);
if(ret <0){
```

```
printf("Program can't be executed\n");
}
}
int status;
while(wait(&status)>0);
}
```

Q13.

```
#include<stdio.h>
#include<stdlib.h>
#include<setjmp.h>
#include<string.h>
int main(int argc, char const *argv[])
{
if(argc != 3){
fprintf(stderr, "Usage : %s <displaymessage> <n>\n", argv[0]);
return 1;
}
int n = atoi(argv[2]);
jmp_buf environment;
setjmp(environment);
if(n == 0){
return 0;
}
n--;
printf("%s\n", argv[1]);
longjmp(environment, n);
return 0;
}
```

Q14.

```
#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>
void main(int argc, char **argv)
{
struct dirent *de;
struct stat fstat;
struct tm *timeinfo;
if(argc != 2){
printf("no month value passed : pass 0-11 month value\n");
exit(1);
}
int month = atoi(argv[1]);
if(month <0 || month > 11){
printf("invalid month value : pass 0-11 month value\n");
exit(1);
}
DIR *directory = opendir(".");
char **filenames;
if (directory == NULL)
{
printf("Could not open current directory" );
return;
}
while ((de = readdir(directory)) != NULL)
if(strcmp(de->d_name,".") != 0 && strcmp(de->d_name,"..")){
stat(de->d_name,&fstat);
timeinfo = localtime(&fstat.st_ctime);
```

```
if(timeinfo->tm_mon == month){
printf("%s\n", de->d_name);
}
}
closedir(directory);
}
```

Q15.

```
#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 %I:%M %p", modified_time);
printf("Modified time : %s\n", timestr);
struct tm *access_time = localtime(&st.st_atime);
strftime(timestr, 80, "%b %d %I:%M %p", access_time);
printf("Access time : %s\n", timestr);
return 0;
}
```

Q16.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/wait.h>
#include<errno.h>
void main(){
```

```
int filedes[2];
if (pipe(filedes) == -1) {
perror("pipe");
exit(1);
}
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);
}
```

Q17.

```
#include<stdio.h>
#include<stdlib.h>
void main() {
FILE *fp1, *fp2;
char ch;
fp1 = fopen("file1.txt", "r");
fp2 = fopen("file2.txt", "w");
while ((ch = fgetc(fp1)) != EOF) {
putc(ch, fp2);
}
printf("File copied Successfully!");
fclose(fp1);
fclose(fp2);
remove("file1.txt");
}
```

Q18.

```
#include<stdio.h>
#include<signal.h>
#include<unistd.h>
#include<malloc.h>
#include<stdlib.h>
#include<string.h>
#include<sys/wait.h>
int signalReceived = 0;
int sigid = 0;
void child_process(char **arguments){
if((execvp(arguments[0], arguments)) < 0){
fprintf(stderr, "error running command : %s\n", arguments[0]);
exit(1);
}
}
```

```
void sigalarm(int sig){
signalReceived = 1;
sigid = SIGALRM;
}
void sigchild(int sig){
signalReceived = 1;
sigid = SIGCHLD;
}
int main(int argc, char *argv[])
{
if(argc < 2){
fprintf(stderr, "Usage : %s <command> [arguments]\n", argv[0]);
return 1;
}
int pid = fork();
int alarmCounter = 0;
if(pid < 0){
fprintf(stderr, "Error in fork execution\n");
return 1;
}
if(pid == 0){
child_process(&argv[1]);
exit(0);
}
signal(SIGCHLD, sigchild);
signal(SIGALRM, sigalarm);
alarm(1);
while(1){
if(signalReceived){
signalReceived = 0;
if(sigid == SIGALRM){
```

```
alarmCounter++;
if(alarmCounter < 5){
alarm(1);
continue;
}
}
if(alarmCounter >= 5){
kill(pid, SIGKILL);
break;
}
if(sigid == SIGCHLD){
signal(SIGALRM, SIG_DFL);
return 0;
}
}
}
return 0;
}
```

Q19.

```
#include<stdio.h>
#include<sys/wait.h>
#include<stdlib.h>
#include<unistd.h>
void child_process(){
printf("Hello World\n");
}
void main(){
int pid = fork();
if(pid < 0){
fprintf(stderr, "Error in fork execution\n");
exit(1);
```

```
}
if(pid == 0){
child_process();
exit(0);
}
int status;
while(wait(&status)>0);
printf("Child exited with status : %d\n", WEXITSTATUS(status));
}
```

Q20.

```
#include<stdio.h>
#include<dirent.h>
#include<stdlib.h>
#include<string.h>
#include<sys/stat.h>
#include<unistd.h>
#include<sys/wait.h>
void main(){
DIR *dir = opendir(".");
struct dirent *dent;
mkdir("./backup", 0777);
char tofilename[256];
while((dent = readdir(dir)) != NULL){
if(dent->d_type == DT_REG){
printf("copying file : %s\n", dent->d_name);
if(fork()==0){
strcat(strcat(tofilename,"./backup/"),dent->d_name);
printf("to %s\n", tofilename);
char* args[] = {"cp",dent->d_name,tofilename, NULL};
execvp("cp",args);
exit(0);
```

```
}
}
}
int status;
while(wait(&status)>0);
}
```