# USP Lab | CIE

Suhas Katrahalli

v1.1

1.

i) Write a C program

a. to read first 20 characters from a file

b. seek to 10th byte from the beginning and display 20 characters from there

c. seek 10 bytes ahead from the current file offset and display 20 characters

d. display the file size

ii) Write a C program to illustrate effect of setjmp and longjmp functions on register and volatile variables.

```
i)  Write a C program
a. to read first 20 characters from a file
b. seek to 10th byte from the beginning and display 20 characters from there
c. seek 10 bytes ahead from the current file offset and display 20 characters
d. display the file size

#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h>
int main()
{
  int file=0, n;
  char buffer[25];
  if((file=open("testfile.txt",O_RDONLY)) < -1)
    printf("file open error\n");

  //read first 20 characters from a file
  if(read(file,buffer,20) != 20)
    printf("file read operation failed\n");
  else
    write(STDOUT_FILENO, buffer, 20);
  printf("\n");

  //seek to 10th byte from the beginning and display 20 chars from there
  if(lseek(file,10,SEEK_SET) < 0)
    printf("lseek operation to beginning of file failed\n");
  if(read(file,buffer,20) != 20)
    printf("file read operation failed\n");
  else
    write(STDOUT_FILENO, buffer, 20);
  printf("\n");

  //seek 10 bytes ahead from the current file offset and display 20 chars
  if(lseek(file,10,SEEK_CUR) < 0)
    printf("lseek operation to beginning of file failed\n");
  if(read(file,buffer,20) != 20)
    printf("file read operation failed\n");
  else
    write(STDOUT_FILENO, buffer, 20);
  printf("\n");
```

```
  //display the file size
  if((n = lseek(file,0,SEEK_END)) <0){
    printf("lseek operation to end of file failed\n");
  }
  printf("size of file is %d bytes\n",n);
  close(file);
  return 0;
}

ii) Write a C program to illustrate effect of setjmp and longjmp functions on register and volatile variables.

//Volatile tells the compiler not to optimize access to the variable
//Register suggests that the variable should be stored in a CPU register if possible.
//register variables are stored in CPU registers, which are not preserved by setjmp() and longjmp().
//volatile variables is preserved across the jump, since it is marked as volatile.
//This ensures that the compiler does not optimize away the variable and that its value is always read from memory.

#include <stdio.h>
#include <setjmp.h>

jmp_buf buf;

int main() {
    volatile int volatile_var = 10;
    register int register_var = 20;

    printf("Initial values: volatile_var = %d, register_var = %d\n", volatile_var, register_var);

    if (setjmp(buf)) {
        // This code will be executed if longjmp is called
        printf("After longjmp: volatile_var = %d, register_var = %d\n", volatile_var, register_var);
        return 0;
    }

    // This code will be executed on first call to setjmp
    volatile_var = 30;
    register_var = 40;

    printf("After setting new values: volatile_var = %d, register_var = %d\n", volatile_var, register_var);

    longjmp(buf, 1);
}
OUTPUT
Initial values: volatile_var = 10, register_var = 20
After setting new values: volatile_var = 30, register_var = 40
After longjmp: volatile_var = 10, register_var = 20
```

💡 2. Write a C program which takes file descriptor as an argument and prints the description of selected file flags for that descriptor.

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
int
main(int argc, char *argv[])
{
  int val;
  if (argc != 2)
    printf("usage: a.out <descriptor#>");
  if ((val = fcntl(atoi(argv[1]), F_GETFL, 0)) < 0)
    printf("fcntl error for fd %d", atoi(argv[1]));
  switch (val & O_ACCMODE) {
    case O_RDONLY:
      printf("read only");
      break;
    case O_WRONLY:
```

```
      printf("write only");
      break;
    case O_RDWR:
      printf("read write");
      break;
    default:
      printf("unknown access mode");
    }
    if (val & O_APPEND)
      printf(", append");
    if (val & O_NONBLOCK)
      printf(", nonblocking");
    if (val & O_SYNC)
      printf(", synchronous writes");
    putchar('\n');
    exit(0);
}

gcc ques2.c -o fd
$ ./fd 0 < /dev/tty
read only
$ ./fd 1 > temp.foo
$ cat temp.foo
write only
$ ./fd 2 2>>temp.foo
write only, append
$ ./fd 5 5<>temp.foo
read write
```

💡 3. Write a C program to simulate system function.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]){
  int status;
  if (argc < 2)
    printf("command-line argument required");

  if ((status = system(argv[1])) < 0)
    printf("system() error");

  printf(status);
  exit(0);
}

#include<stdio.h>

int main(){
  printf("Hello");
}

OUTPUT: We are executing the command-line argument using system function
$ gcc ques3.c -o ques3
$ gcc testf.c -o testf
$ ./ques3 ./testf
```

💡 4. 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<sys/types.h>
#include<fcntl.h>
#include<sys/wait.h>
int main(void)
{
    pid_t pid;
    off_t offset;
    char buffer[32];
    int fd, status;
    if((fd = open("test2.txt",O_CREAT | O_RDWR)) == -1) {
        printf("Read error\n");
        exit(1);
    }
    write(fd,"Hi abc from msrit\n",18);
    pid = fork();
    if(pid == -1) {
        printf("Fork error\n");
        exit(1);
    }
    else if(pid == 0) {
        offset = lseek(fd,0,SEEK_CUR);
        printf("Child current offset %ld\n",offset);
        lseek(fd,0,SEEK_SET);

        read(fd,buffer,14);
        printf("%s\n", buffer);

        lseek(fd,5,SEEK_SET);
        printf("Child's current offset is %ld\n",lseek(fd,0,SEEK_CUR));
    }
    else {
        wait(&status);
        offset = lseek(fd,0,SEEK_CUR);
        printf("Parent current offset %ld\n",offset);
        lseek(fd,0,SEEK_CUR);

        read(fd,buffer,5);
        printf("%s\n", buffer);


        printf("Parent's current offset %ld\n",lseek(fd,0,SEEK_CUR));
    }
    return 0;
}
OUTPUT
Child current offset 18
Hi abc from ms
Child's current offset is 5
Parent current offset 5
c fro
Parent's current offset 10
```

💡 5. Write a C program to copy access and modification time of a file to another file using utime function.

```
check
#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));
}


#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <utime.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include<unistd.h>
int main(int argc, char *argv[])
{
int i, fd;
struct stat statbuf;
struct utimbuf timebuf;
for (i = 1; i < argc; i++) {
if (stat(argv[i], &statbuf) < 0) { /* fetch current times */
printf("%s: stat error", argv[i]);
continue;
}
if ((fd = open(argv[i], O_RDWR | O_TRUNC)) < 0) { /* truncate */
printf ("%s: open error", argv[i]);
continue;
}
close(fd);
timebuf.actime = statbuf.st_atime;
timebuf.modtime = statbuf.st_mtime;
if (utime(argv[i], &timebuf) < 0)
{ /* reset times */
printf("%s: utime error", argv[i]);
continue;
}
printf("%ld %ld",timebuf.actime,timebuf.modtime);
}
exit(0);
}

#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <utime.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/stat.h>
int main(int argc, char *argv[])
{
int i, fd;
struct stat statbuf;
struct utimbuf timebuf;
for (i = 1; i < argc; i++) {
if (stat(argv[i], &statbuf) < 0) { /* fetch current times */
printf("%s: stat error", argv[i]);
```

```
continue;
}
if ((fd = open(argv[i], O_RDWR | O_TRUNC)) < 0) { /* truncate */
printf ("%s: open error", argv[i]);
continue;
}

close(fd);
timebuf.actime = statbuf.st_atime;
timebuf.modtime = statbuf.st_mtime;

if (utime(argv[i], &timebuf) < 0)
{ /* reset times */
printf("%s: utime error", argv[i]);
continue;
}
}
exit(0);
}

Output:
gedit prog1.c
cc prog1.c -o prog1
./prog1 lab4.txt foo.html //give two filenames as command line args
Before Copying ...
Access Time Thu Apr 19 15:04:46 2018

Modification Time Thu Apr 19 15:04:46 2018

After Copying ...
Access Time Tue Feb 20 12:41:11 2018

Modification Time Tue Feb 20 12:41:11 2018

//here lab4.txt's time is changed to foo.html's time
```

💡 6. Write a C program to perform the following operations
a. To create a child process
b. Child process should execute a program to show the use of the access function
c. Parent process should wait for the child process to exit
d. Also print the necessary process IDs

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <fcntl.h>

int main() {
    pid_t pid;

    // create a child process
    pid = fork();

    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    } else if (pid == 0) {
        // child process executes a program to show the use of the access function
        if (access("test.txt", F_OK) == 0) {
            printf("test.txt exists and is readable\n");
        } else {
```

```
            printf("test.txt does not exist or is not readable\n");
        }
        exit(EXIT_SUCCESS);
    } else {
        // parent process waits for the child process to exit
        int status;
        waitpid(pid, &status, 0);
        printf("Parent process: Child process exited with status %d\n", status);
        printf("Parent process ID: %d\n", getpid());
        printf("Child process ID: %d\n", pid);
    }

    return 0;
}
```

💡 7. Write a C program to demonstrate race condition in UNIX environment and provide the solution for the same

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/stat.h>

static int pfd1[2],pfd2[2];
static void charatatime(char *);
 void  WAIT_PARENT();
 void TELL_CHILD(pid_t);
 void TELL_WAIT();

int main(void)
{
pid_t pid;
TELL_WAIT();
if ((pid = fork()) < 0) {
printf("fork error");
exit(1);
} else if (pid == 0) {
WAIT_PARENT();
charatatime("output from child\n");
} else {
charatatime("output from parent\n");
TELL_CHILD(pid);
}
exit(0);
}

static void charatatime(char *str)
{

char *ptr;
int c;
setbuf(stdout, NULL);
/* set unbuffered */
for (ptr = str; (c = *ptr++) != 0; )
putc(c, stdout);
}
void WAIT_PARENT()
{
char c;
if (read(pfd1[0], &c, 1) != 1)
{
    printf("read error");
    exit(1);
```

```
}
if (c != 'p')
{
    printf("WAIT_PARENT: incorrect data");
    exit(1);
}
}
void TELL_CHILD(pid_t pid)
{
if (write(pfd1[1], "p", 1) != 1)
{
    printf("write error");
    exit(1);
}
}
void TELL_WAIT(void)
{
if (pipe(pfd1) < 0 || pipe(pfd2) < 0)
{printf("pipe error");
exit(1);}
}
```

💡 8. Write a C program to avoid zombie status of a process. Justify the output

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid = fork();

    if (pid < 0) {
        perror("fork failed");
        exit(1);
    } else if (pid == 0) {
        // child process
        printf("Child process with PID %d is sleeping\n", getpid());
        sleep(5);
        printf("Child process with PID %d is done sleeping\n", getpid());
        exit(0);
    } else {
        // parent process
        printf("Parent process with PID %d is waiting for child with PID %d\n", getpid(), pid);
        wait(NULL);
        printf("Parent process with PID %d is done waiting\n", getpid());
        exit(0);
    }
}
```

💡 2 or

```
#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);
}

output:
gedit prog1.c
cc prog1.c -o prog1
./prog1
//output:
in f1():
globval = 95, autoval = 96, regival = 97, volaval = 98, statval = 99
after longjmp:
globval = 10000, autoval = 96, regival = 97, volaval = 98, statval = 99
```

💡 3 or

```
#include<errno.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdio.h>
int system1(const char *cmdstring)
{
  pid_t pid;
  int status;
  if(cmdstring == NULL)
    return(1);
  pid=fork();
  if(pid< 0)
    status = -1;
  else if(pid == 0)
  {
    execl("/bin/sh","sh","-c",cmdstring,(char *)0);
    _exit(127);
  }
  else
  {
    while(waitpid(pid,&status,0) < 0)
```

```
    {
      if(errno != EINTR)
      {
        status = -1;
        break;
      }
    }
  }
  return(status);
}
int main()
{
  int status;
  status = system1("ls -ls");
  printf("Command executed with status %d",status);
  status = system1("date");
  printf("Command executed with status %d",status);
  status = system1("sdfsd");
  printf("Command executed with status %d",status);
}
```

💡 4 or

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

int main() {
    int fd;
    pid_t pid;

    fd = open("file.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
    if (fd == -1) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    pid = fork();
    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        // Child process
        printf("Child process: fd = %d\n", fd);

        // Write to the file at position 0
        lseek(fd, 0, SEEK_SET);
        write(fd, "Child process writes to file\n", 30);

        // Read from the file at position 31
        char buffer[30];
        lseek(fd, 31, SEEK_SET);
        read(fd, buffer, 30);
        printf("Child process read from file: %s\n", buffer);

        // Close the file descriptor
        close(fd);
        exit(EXIT_SUCCESS);
    } else {
        // Parent process
        printf("Parent process: fd = %d\n", fd);
```

```
        // Write to the file at position 31
        lseek(fd, 31, SEEK_SET);
        write(fd, "Parent process writes to file\n", 31);

        // Wait for the child to exit
        wait(NULL);

        // Read from the file at position 0
        char buffer[31];
        lseek(fd, 0, SEEK_SET);
        read(fd, buffer, 31);
        printf("Parent process read from file: %s\n", buffer);

        // Close the file descriptor
        close(fd);
        exit(EXIT_SUCCESS);
    }
}

OUTPUT
Parent process: fd = 3
Child process: fd = 3
Parent process read from file: Child process writes to file
Child process read from file: Parent process writes to file
```

💡 8 or

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main(void) {
    pid_t pid;
    if ((pid = fork())< 0) {
        printf("forkerror");
    }
    else if (pid == 0) { /* first child */
        if((pid = fork()) < 0)
            printf("fork error");
        else if (pid > 0)
            exit(0);
        sleep(2);
        printf("second child, parent pid = %ld\n",(long)getppid());
        exit(0);
}
if (waitpid(pid, NULL, 0) != pid)
    printf("waitpid error");
exit(0);
}
```