Assignment-2

① Program to illustrate Access System call.

Ans) **access**: The access system call checks the existence & access permission of user to a named file.

Prototype: #include<unistd.h>
int access(const char *path_name, int flag);

Ex: C program to check access permissions of the file using access system call.

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<unistd.h>
#include<sys/types.h>
int main(int argc, char *argv[])
{
 if(argc!=2)
  printf("usage: argv[0] <pathname>");
 if(access(argv[1], R_OK)<0)
  printf("access error for %s\n", argv[1]);
else
  printf("read access is allowed\n");
if(open(argv[1], O_RDONLY)<0)
  printf("opening a file is error because the file has
        permissions of write, read-write for %s\n",argv[1]);
  else
```

```
printf("Access permission for the file is read-only\n");
exit(0);
}
```

OUTPUT

1) ./a.out access.txt

read access is allowed
Access permission for the file is read-only

2) ./a.out stat.txt

access error for stat.txt
opening a file is error because the file has permissions of write, read-write for stat.txt.

---

② Program to illustrate atexit() function.

Ans2 atexit(): This declaration says that we pass the address of a function as the argument to atexit. When this function is called, it is not passed any arguments & is not expected to return a value. The exit function calls these functions in reverse order of their registration. Each function is called as many times as it was registered.

Prototype: #include<Stdlib.h>
int atexit(void(*func)(void));

C program to illustrate atexit() API.

```
#include<Stdio.h>
#include<Stdlib.h>
int i;
void check_a(void)
{
 i++;
 printf("Check A: %d\n", i);
}
```

```
Void check_b(void)
{
  i++;
  Printf("Check B: %d\n", i);
}
int main()
{
 check_a();
atexit(check_a);
 check_b();
 printf("Exiting.\n");
  return 0;
}
```

OUTPUT

```
./a.out
Check A: 1
Check B: 2
Exiting.
Check B: 3
Check A: 4.
```

Program to illustrate Vfork() function.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
 int glob=6;
 int main(void) {
```

```
int var;
pid_t pid;
var = 88;
printf("before vfork\n");
if((pid = vfork()) < 0)
{
perror("vfork error\n");
}
else if(pid == 0)
{
glob++;
exit(0);
}
printf("pid = %d, glob = %d, var = %d\n", getpid(),
        glob, var);
exit(0);
}
```

OUTPUT

```
before vfork
pid = 4648, glob = 7, var = 89.
```

---

④ Program to illustrate race condition.

Ans

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
static void charatatime(char *);
int main(void)
{
```

```
pid_t pid;
if((pid = fork()) < 0)
{
 printf("fork error");
}
else if (pid == 0)
{
charatatime("output from child");
}
else
{
charatatime("output from parent");
}
exit(0);
}
Static void charatatime(char *str)
{
 char *ptr;
 int c;
setbuf(stdout, NULL);
for(ptr = str; (c = *ptr++) != 0;)
 putc(c, stdout);
}
```

OUTPUT

```
./a.out
output from child output
 from parent.
```

(5) Program to avoid Zombie process by calling fork twice.

Ans 5

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
int main()
{
 pid_t pid;
 pid=fork();
 if(pid==0)
{
  pid=fork();
  if(pid>0)
{
 exit(0);
}
 sleep(2);
 printf("Second child parent pid=%d", getpid());
 exit(0);
}
if(waitpid(pid, NULL, 0)!=pid)
{
 printf("error");
}
 exit(0);
}
```

OUTPUT

Second child parent pid = 1761

⑥ Program to illustrate exec() function.

Ans

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
char *env_init[]={"USER=unknown", "PATH=/tmp",
                  NULL};
int main(void)
{
pid_t pid;
if((pid=fork())<0)
{
 printf("fork error");
}
else if (pid==0)
{
if (execle("/home/sar/bin/echoall", "echoall", "myarg1",
     "MY ARG2", (char *)0, env_init)<0)
 printf("execle error");
}
if(waitpid(pid, NULL, 0)<0)
Printf("wait error");
 exit(0);
}
```

OUTPUT

```
./a.out
argv[0]: echoall
argv[1]: myarg1
argv[2]: MY ARG2
USER: unknown
PATH=/tmp
$ argv[0]: echoall
argv[1]: only 1 arg
USER=sar
LOGNAME=sar
SHELL=/bin/bash
HOME=/home/sar.
```