USN											CMRIT

Internal Assessment Test 3 –JAN 2023
Scheme of Evaluation

Sub:	**UNIX Programming**					Sub Code:	18CS56	Branch:	ISE
Date:	20/01/2023	Duration:	90 min	Max Marks:	50	Sem/Sec:	V / A, B & C		**OBE**

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	Explain signal mask with an example. Scheme: Definition+Explanation of prototypes+example- 3+4+3 marks Solution:	[10]	CO5	L2

- ✓Each process in UNIX or POSIX.1 system has signal mask that defines which signals are blocked when generated to a process.
- ✓A blocked signal depends on the recipient process to unblock it and handle it accordingly.

A process may query or set its signal mask via the sigprocmask API:

```
#include <signal.h>
int sigprocmask(int cmd, const sigset_t *new_mask, sigset_t *old_mask);
```

```
#include <signal.h>
int sigemptyset(sigset_t *sigmask);
int sigaddset(sigset_t *sigmask, const int signal_num);
int sigdelset(sigset_t *sigmask, const int signal_num);
int sigfillset(sigset_t sigmask);
int sigismember(const sigset_t *sigmask, const int signal_num);
```

```
#include <stdio.h>
#include <signal.h>
int main()
{
sigset_t sigmask;
sigemptyset(&sigmask); /*initialize set*/
if (sigprocmask(0, 0, &mask) == -1) { /*get current signal mask*/
perror("sigprocmask");
exit(1);
} else
sigaddset(&sigmask, SIGINT); /*set SIGINT flag*/
sigdelset(&sigmask, SIGSEGV); /*clear SIGSEGV flag*/
if (sigprocmask(SIG_SETMASK, &sigmask, 0) == -1)
perror("sigprocmask"); /*set a new signal mask*/
}
```

2	Define signal and Discuss any 10 POSIX-defined signals in UNIX with example. Scheme:Definition+POSIX Defined Signals+Example- 3+4+3 marks Solution:	[10]	CO5	L1

✓Signals are triggered by events and are posted on a process to notify it that something has happened and requires some action.

✓Signals can be generated from a process, a user, or the UNIX kernel.

Example:-

a. A process performs a divide by zero or dereferences a NULL pointer.

b. A user hits <Delete> or <Ctrl-C> key at the keyboard.

Name	Description Default action
SIGALRM	timer expired (alarm) terminate
SIGABRT	abnormal termination (abort) terminate+core
SIGFPE	arithmetic exception terminate+core
SIGHUP	controlling terminal hangup terminate
SIGILL	illegal machine instruction terminate+core
SIGINT	terminal interrupt character <delete> or <ctrl-c> keys terminate
SIGKILL	kill a process, kill -9 <pid> command. terminate
SIGPIPE	write to pipe with no readers terminate
SIGQUIT	terminal quit character terminate+core
SIGSEGV	segmentation fault - invalid memory reference terminate+core
SIGTERM	terminate process, kill <pid> command terminate

```
#include <signal.h>
void (*signal (int signal_num, void (*handler)(int))(int);
```

```
#include <iostream.h>
#include <signal.h>
void catch_sig(int sig_num) // Signal handler function
{
        signal(sig_sum, catch_sig);
        cout << “catch_sig:” << sig_num << endl;
}
int main()
{
        signal(SIGTERM, catch_sig);
        signal(SIGINT, SIG_IGN);
        signal(SIGSEGV, SIG_DFL);
        pause(); // wait for signal interruption
}
```

3	Explain IPC mechanism of Message queues along with API Functions. Scheme:Defn &msqid+Prototypes of msg queues with explanation- 5+5 marks	[10]	CO4	L2

Solution:

- A message queue is a linked list of messages stored within the kernel and identified by a message queue .
- A new queue is created or an existing queue opened by **msgget**.
- New messages are added to the end of a queue by **msgsnd**.
- Messages are fetched from a queue by **msgrcv**.
- Each queue has the following msqid_ds structure associated with it:

```
struct msqid_ds {
  struct ipc_perm  msg_perm;     /* see Section 15.6.2 */
  msgqnum_t        msg_qnum;     /* # of messages on queue */
  msglen_t         msg_qbytes;   /* max # of bytes on queue */
  pid_t            msg_lspid;    /* pid of last msgsnd() */
  pid_t            msg_lrpid;    /* pid of last msgrcv() */
  time_t           msg_stime;    /* last-msgsnd() time */
  time_t           msg_rtime;    /* last-msgrcv() time */
  time_t           msg_ctime;    /* last-change time */
   :
};
```

```
#include <sys/msg.h>
int msgget(key_t key, int flag);
```
Returns: message queue ID if OK, 1 on error.

```
#include <sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buf );
```
Returns: 0 if OK, 1 on error.

```
#include <sys/msg.h>
int msgsnd(int msqid, const void *ptr, size_t nbytes, int flag);
```
Returns: 0 if OK, 1 on error.

```
struct mymesg {
  long  mtype;
  char  mtext[512];
};
```

```
#include <sys/msg.h>
ssize_t msgrcv(int msqid, void *ptr, size_t nbytes, long type, int flag);
```
Returns: size of data portion of message if OK, 1 on error.

4 (a) Write a C Program in calling a system function for process control. [05] CO4 L3

Scheme: Program – 5 Marks

Solution:

Calling the system function

```c
#include "apue.h"
#include <sys/wait.h>

int
main(void)
{
    int     status;

    if ((status = system("date")) < 0)
        err_sys("system() error");

    pr_exit(status);

    if ((status = system("nosuchcommand")) < 0)
        err_sys("system() error");

    pr_exit(status);

    if ((status = system("who; exit 44")) < 0)
        err_sys("system() error");

    pr_exit(status);

    exit(0);
}
```

(b) Write a C program to generate accounting data of a process. [05] CO4 L3

Scheme: Program – 5 Marks

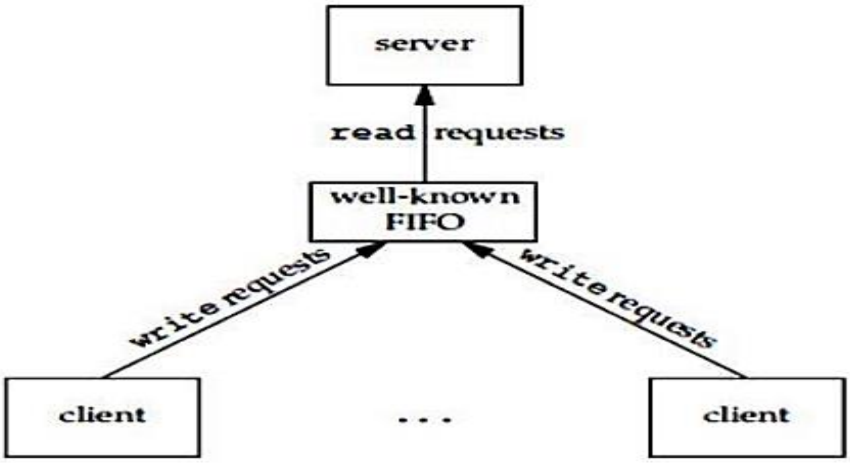
Solution:

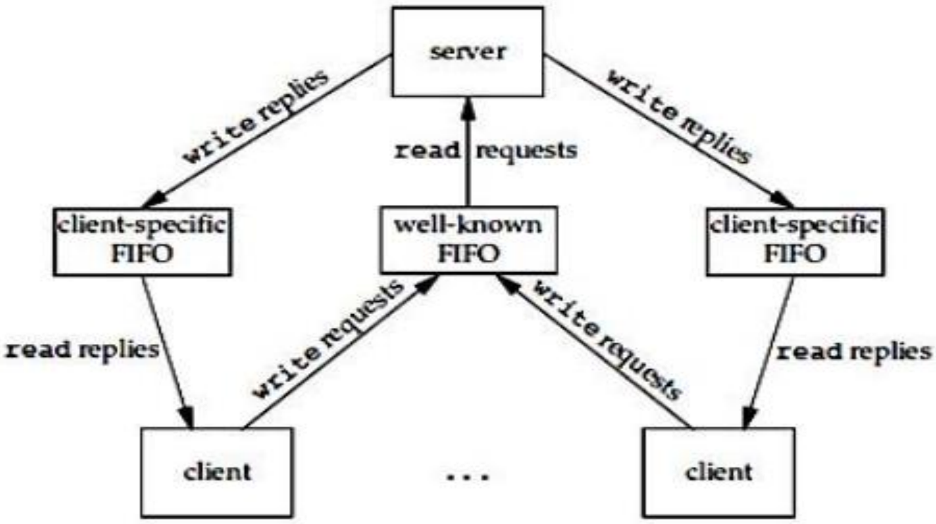
```c
#include "apue.h"

int
main(void)
{
    pid_t   pid;

    if ((pid = fork()) < 0)
        err_sys("fork error");
    else if (pid != 0) {        /* parent */
        sleep(2);
        exit(2);                /* terminate with exit status 2 */
    }
    if ((pid = fork()) < 0)
        err_sys("fork error");
    else if (pid != 0) {        /* first child */
        sleep(4);
        abort();                /* terminate with core dump */
    }
    if ((pid = fork()) < 0)
        err_sys("fork error");
    else if (pid != 0) {        /* second child */
        execl("/bin/dd", "dd", "if=/etc/passwd", "of=/dev/null", NULL);
        exit(7);                /* shouldn't get here */
    }
    if ((pid = fork()) < 0)
        err_sys("fork error");
    else if (pid != 0) {        /* third child */
        sleep(8);
        exit(0);                /* normal exit */
    }
    sleep(6);                   /* fourth child */
    kill(getpid(), SIGKILL);    /* terminate w/signal, no core dump */
    exit(6);                    /* shouldn't get here */
}
```

<table>
<tr>
<td>5</td>
<td>
Define FIFO IPC and Explain Client/Server Communication using FIFO with a neat diagram.
Scheme: Definition with Prototypes+Client/Server Communication – 3+3+4 marks
Solution:

```
#include <sys/stat.h>
int mkfifo(const char *path, mode_t mode);
int mkfifoat(int fd, const char *path, mode_t mode);
Both return: 0 if OK, -1 on error.
```

- If the path parameter specifies an absolute pathname, then the fd parameter is ignored and the mkfifoat function behaves like the mkfifo function.
- If the path parameter specifies a relative pathname and the fd parameter is a valid file descriptor for an open directory, the pathname is evaluated relative to this directory
- FIFOs are used by shell commands to pass data from one shell pipeline to another without creating intermediate temporary files.
- FIFOs are used in client-server applications to pass data between the clients and the servers.





</td>
<td>[10]</td>
<td>CO4</td>
<td>L2</td>
</tr>
</table>

6 (a)	Explain Single Instance Daemons and Daemon Characteristics. Scheme: Explanation of Single Instance+ Daemon Characteristics-2+3 Marks Solution: • Some daemons are implemented so that only a single copy of the daemon should be running at a time for proper operation. • The file and record-locking mechanism provides the basis for one way to ensure that only one copy of a daemon is running. • If each daemon creates a file and places a write lock on the entire file, only one such write lock will be allowed to be created. • Successive attempts to create write locks will fail, serving as an indication to successive copies of the daemon that another instance is already running. Daemons run in background. Daemons have super-user privilege. Daemons don't have controlling terminal. Daemons are session and group leaders.	[05]	CO5	L1
(b)	Explain UNIX Kernel Support for Signals. Scheme: Kernel support explanation-5 Marks Solution: 1. In Unix System V.3, each entry in the kernel process table slot has an array of signal flags, one for each defined in the system. 2. When a signal is generated for a process, the kernel will set the corresponding signal flag in the process table slot of the recipient process. 3. If the recipient process is asleep (waiting a child to terminate or executing *pause* API) the kernel will awaken the process by scheduling it. 4. When the recipient process runs, the kernel will check the process U-area that contains an array of signal handling specifications, where each entry of the array corresponds to a signal defined in the system. 5. The kernel will consult the array to find out how the process will react to the pending signal.	[05]	CO5	L1

Faculty Signature CCI Signature HOD Signature