1.

Client.c:

NAME : SHREERAM

REG NO: 20214033

GROUP: D

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
 int main() {     int num1, num2, result;     char
operator;     char num1_str[10], num2_str[10];
    while (1) {         printf("Enter two integers and an operator (+ or -), separated by spaces: ");
scanf("%d %d %c", &num1, &num2, &operator);
       // Create a child process         pid_t pid = fork();
        if (pid < 0) {             perror("Fork
failed");             exit(1);
       } else if (pid == 0) {
          // Child process (server)

          // Convert integers to strings             snprintf(num1_str, sizeof(num1_str),
"%d", num1);             snprintf(num2_str, sizeof(num2_str), "%d", num2);
          // Execute the server program with execl             execl("./server", "server", num1_str, num2_str,
&operator, NULL);
          // If execl fails             perror("execl");
exit(1);
       } else {
          // Parent process (client)
```

```c
          wait(NULL);
           printf("Ready for a new reading.\n");
       }
   }      return 0;
}
```

Server.c:

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

int main(int argc, char *argv[]) {     if (argc != 4) {         fprintf(stderr, "Usage: %s <num1> <num2>
<operation>\n", argv[0]);         return 1;
    }       int num1 = atoi(argv[1]);     int
num2 = atoi(argv[2]);     char operation =
argv[3][0];
     if (operation == '+') {         int result =
num1 + num2;
        printf("Server process: Result = %d\n", result);         return result;
    } else if (operation == '-') {         int result =
num1 - num2;
        printf("Server process: Result = %d\n", result);         return result;
    } else {         printf("Server process: Invalid operation\n");         return
1;
    }
}
```

2.

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

```
#include <sys/wait.h>
 int main() {     int total_processes =
0;
    for (int i = 1; i <= 10; i++) {         pid_t
child_pid = fork();
        if (child_pid == 0) {                               // Child process             printf("Child process
with PID=%d\n", getpid());             exit(0);
        } else if (child_pid > 0) {                          // Parent process             total_processes++;
        } else {
           perror("Fork failed");             exit(1);
        }
    }

   // Parent process waits for all child processes to finish     for (int i = 0; i <
total_processes; i++) {         wait(NULL);
    }

   FILE *file = fopen("process_management.txt", "w");     if (file != NULL) {         fprintf(file,
"Total processes created: %d\n", (1 << 10) - 1);         fclose(file);
    } else {
       perror("Failed to open file");
    }      return 0;
}
```

3.

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>  void sigint_handler(int
signum) {     printf("\nSignal %d (SIGINT)
received. Exiting...\n", signum);
```

```
    exit(0);
}  int main() {     if (signal(SIGINT, sigint_handler) == SIG_ERR) {
perror("Signal registration failed");         return 1;
    }      while (1) {         printf("Running. Press Ctrl+C to trigger SIGINT.\n");
sleep(1);
    }      return 0;
}
```

4.

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

volatile int g_signal_received = 0;
 void signal_handler(int signal) {     if (signal == SIGINT) {
g_signal_received = 1;         printf("SIGINT received. Unblocking
signals.\n");
    }
}  int main() {     struct sigaction sa;
sa.sa_handler = signal_handler;     sa.sa_flags
= 0;
    sigemptyset(&sa.sa_mask);
     if (sigaction(SIGINT, &sa, NULL) == -1) {
perror("sigaction");         return 1;
    }
    printf("Press Ctrl+C to unblock signals.\n");
```

```
   while (1) {         if (g_signal_received) {
          // Unblock signals here as needed
          // For simplicity, we unblock all signals here              sigset_t all_signals;
sigfillset(&all_signals);             sigprocmask(SIG_UNBLOCK, &all_signals,
NULL);             break;
       }
       sleep(1);
   }       while (1) {
       // Signals are unblocked at this point
printf("Hello\n");         sleep(1);
   }       return 0;
}
```