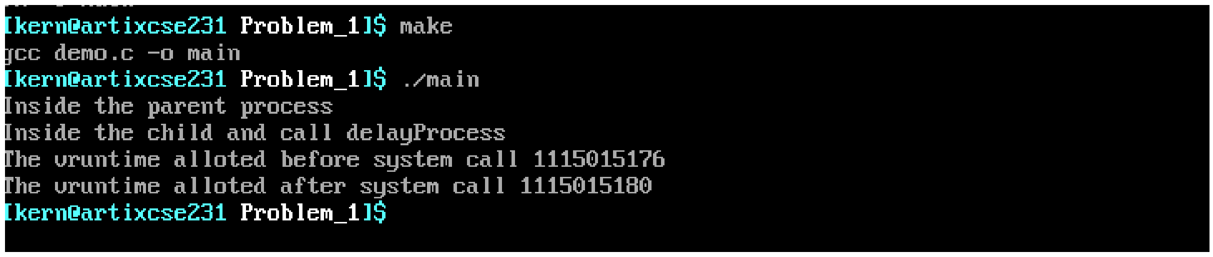
Directory Structure

- **Assignment_2**
  - **Problem_1** - Implementation of system call for delaying the current process
    - patch - The diff file with the Linux-5.14.3 and modified kernel.
    - demo.c - The demo program for testing the system call
    - changes.txt - It contains the changes made. Just for reference purposes.
    - makefile
  - **Problem_2_Fifo** - Implementation of Problem 2 using FIFO
    - P1.c- Program 1 file using FIFO
    - P2.c - Program 2 file using FIFO
    - main.c - A main program to launch the above two P1.c and P2.c executable using exec system call.
    - makefile
  - **Problem_2_MessageQueue** - Implementation of Problem 2 using message queues.
    - P1.c- Program 1 file using message queues.
    - P2.c - Program 2 file using message queues.
    - main.c - main program to launch the above two P1.c and P2.c executable using exec system call.
    - makefile
  - **Problem_2_Socket** - Implementation of Problem 2 using Unix Domain Sockets.
    - P1.c- Program 1 file using Domain Sockets
    - P2.c - Program 2 file using Domain Sockets
    - main.c - A main program to launch the above two P1.c and P2.c executable using exec system call.
    - makefile
  - readme.pdf

Problem_1

Input
No input needs to be given for running the program.

Output
The allotted vruntime for the current process before the system call and after the system call would be printed on the screen.

Expected Output


```
[kern@artixcse231 Problem_1]$ make
gcc demo.c -o main
[kern@artixcse231 Problem_1]$ ./main
Inside the parent process
Inside the child and call delayProcess
The vruntime alloted before system call 1115015176
The vruntime alloted after system call 1115015180
[kern@artixcse231 Problem_1]$
```

System Calls Used
- syscall - This system call is used to call a particular system call from the syscall table using the syscall number and pass the required arguments to the system call
- **delayProcess(**int delay,int pid,u_int64_t * timeSliceEarlier,u_int64_t * timeSliceAfter,unsigned long dataSize**)** - This is our custom syscall with syscall number 449. It takes 5 arguments as the required delay, pid of the process to delay , timeSliceEarlier Pointer and timeSliceAfter pointer and the Datasize of the timeslice pointers.

Compiling the Kernel
- First , go to Artix Home Directory and then mkdir Linux-Kernel.
- Then give the command or use the supplied linux-5.14.3 tarball file lynx https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.14.3.tar.xz
- Then extract tarball: tar -xvf linux-5.14.3.tar.xz
- Now put the patch file insider Linux-Kernel directory.
- Now go to the linux source by cd linux-5.14.3
- Now apply patch by the following command
- patch -p1 -R <../patch
- The modification for the custom system call has been added to the kernel source.
- Then paste the supplied .config inside the extracted directory.
- Now we need to compile the kernel using following commands:

- make
- sudo make modules_install
- sudo cp arch/x86_64/boot/bzImage /boot/vmlinuz-linux-5.14.3
- sudo cp System.map System.map-5.14.3
- sudo mkinitcpio -k 5.14.3 -g /boot/initramfs-linux-5.14.3.img
- sudo grub-mkconfig -o /boot/grub/grub.cfg
- sudo reboot

Compiling the Program

- After compiling and configuring the linux kernel do the following
- For compiling the program inside the Problem 2 folder give the command make
- Then run the demo executable using ./main

Source Files modified

- fair.c file located at path linux-5.14.3/kernel/sched/fair.c . We modify this file to add our custom syscall to the CFS scheduler in Linux. It is easy to add the system call in this file as the majority of the related data structures and reference for modifying CFS scheduler like struct sched_entity, cfs_rq are already defined in fair.c
- syscall_64.tbl file located at linux-5.14.3/arch/x86/entry/syscalls/syscall_64.tbl. This file is used to add our system call to the system call table.

Data Structures Used

- u_int64_t - It is used for storing the vruntime of the process
- struct cfs_rq - It is used to access the task tree in our system call.
- struct sched_entity - This is used to access the task struct of the current process.

Description of how the program works

- We need to implement a custom system call.
- So we implement the system call in the fair.c file located at path linux-5.14.3/kernel/sched/fair.c
- We first declare our system call using SYCALL_DEFINE macro.It takes 5 arguments as the required delay, pid of the process to delay , timeSliceEarlier Pointer and timeSliceAfter pointer and the Datasize of the timeslice pointersThe complete instruction is SYSCALL_DEFINE5(delayProcess,int,delay,int,pid,u_int64_t *,timeSliceEarlier,u_int64_t *,timeSliceAfter,unsigned long ,dataSize)
- We also make cfs_rq static in fair.c as cfsStatic so we can access the current in our system call to modify its vruntime according to the delay specified.

- In our system call, we first define struct sched_entity *curr = cfsStatic->curr to get the task struct of the currently running process.
- In our system call we check if the current process has the pid as specified in the system call. If its the same PID , then we get the vruntime of the current process using curr->vruntime and add the required delay to it.
- Now we copy the earlier and modified vruntime using __copy_to_user() to the specified supplied u_int64_t pointers. Then in the demo program we print those earlier and modified vruntime times.

Problem_2_Fifo

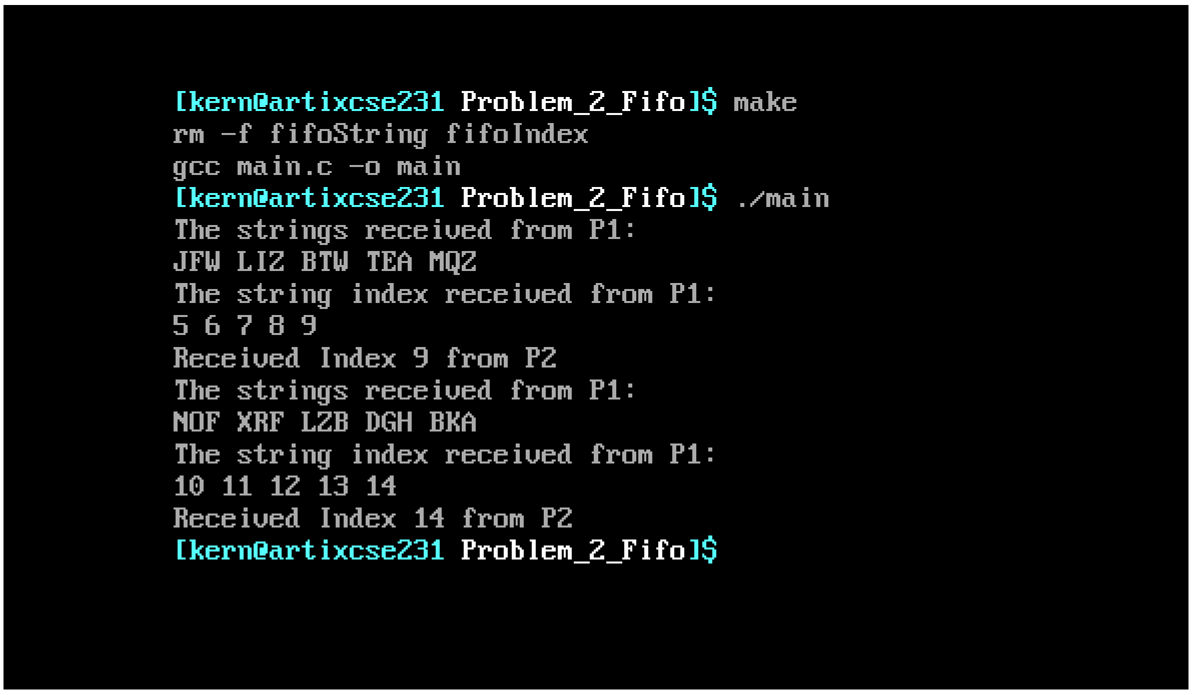
Input

No Input needs to be given. We start the program from index 5 of the random string array

Output

The first five string index starting from 5 would be printed on the string along with the randomly generated string content. Then the highest index would be sent back to P1.P1 now prints the received index 9. P1 now sends back the successor of that index 10, to the P2 which prints the indices and the content. P2 then sends the highest index 14 to the P1. P1 now prints the received index 14.

Expected Output



```
[kern@artixcse231 Problem_2_Fifo]$ make
rm -f fifoString fifoIndex
gcc main.c -o main
[kern@artixcse231 Problem_2_Fifo]$ ./main
The strings received from P1:
JFW LIZ BTW TEA MQZ
The string index received from P1:
5 6 7 8 9
Received Index 9 from P2
The strings received from P1:
NOF XRF LZB DGH BKA
The string index received from P1:
10 11 12 13 14
Received Index 14 from P2
[kern@artixcse231 Problem_2_Fifo]$
```

Since the strings are generated randomly so the values of the strings might be different.

Compiling the Program

For compiling the program a makefile is supplied in the program directory. First, run make in a Linux terminal inside the program folder **Problem_2_Fifo.** Then run the **main** file using **./main**.

Data Structures Used

- char[50][5] - The array for storing random string of length 3.
- int[5] - to store the index of the five strings to be sent.
- struct queueData - A custom structure used to store a long Index and a string of length 5 as mstring.
  ```
  struct queueData
  {
     long index;
     char mString[5];
  }
  ```
-

Logic of the program

- First we generate a character array of size char[50][5].
- Then we traverse through that array and fill the first three rows of each column with a random number generated using rand() which is between 65 and 100, the valid ASCII codes of A-Z.
- Then we select the 5 th index of the array , copy it to an another array toBeSent and an index array that contain the respective index.
- We send these two arrays to P2 using message queue and functions msgrcv and msgsnd to received and send messages respectively.
- P2 reads from those message queues using msgrcv, the strings and indexes required and sends the highest index back to the P1 using msgsnd.

Data Structures Used

- char[50][5] - The array for storing random string of length 3.
- int[5] - to store the index of the five strings to be sent.
- FIFO - These fifo are used in write and read mode by both P1 and P2.

Logic of the program

- First we generate a character array of size char[50][5].
- Then we traverse through that array and fill the first three rows of each column with a random number generated using rand() which is between 65 and 100, the valid ASCII codes of A-Z.
- Then we select the 5 th index of the array , copy it to an another array toBeSent and an index array that contain the respective index.
- We send these two arrays to P2 using 2 FIFO fifoString and fifoIndex respectively.
- P2 reads from those FIFO the strings and indexes required and sends the highest index back to the P1 using FIFO.

Problem_2_MessageQueue

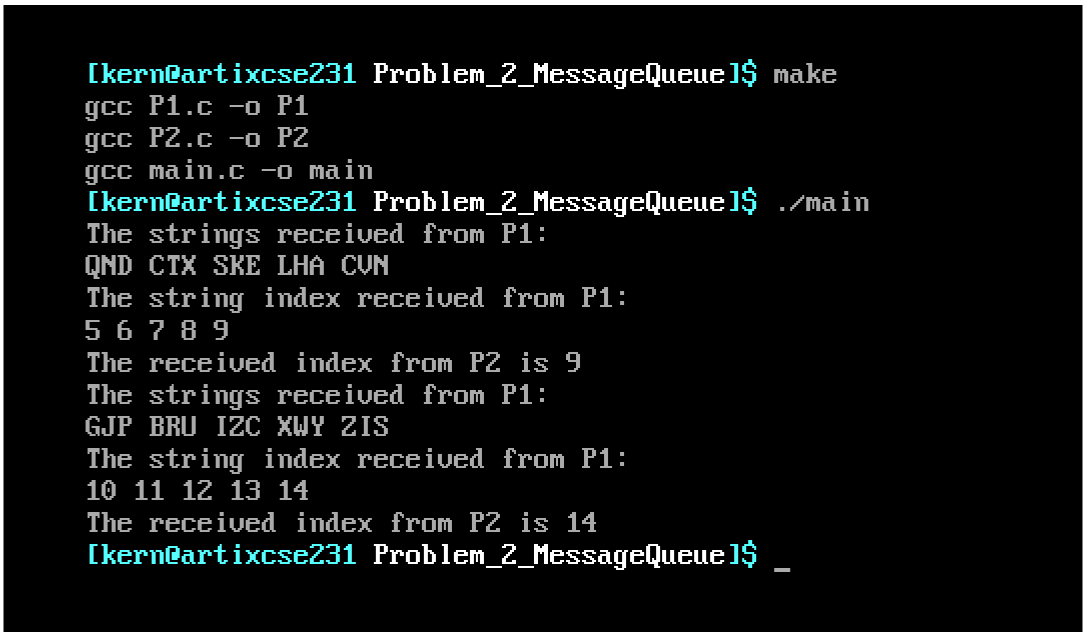
Input

No Input needs to be given. We start the program from index 5 of the random string array. 50 random string of fixed length 3 is created and stored in a string array.

Output

The first five string index starting from 5 would be printed on the string along with the randomly generated string content. Then the highest index would be sent back to P1.P1 now prints the received index 9. P1 now sends back the successor of that index 10, to the P2 which prints the indices and the content. P2 then sends the highest index 14 to the P1. P1 now prints the received index 14.

Expected Output



```
[kern@artixcse23] Problem_2_MessageQueue]$ make
gcc P1.c -o P1
gcc P2.c -o P2
gcc main.c -o main
[kern@artixcse23] Problem_2_MessageQueue]$ ./main
The strings received from P1:
QND CTX SKE LHA CVN
The string index received from P1:
5 6 7 8 9
The received index from P2 is 9
The strings received from P1:
GJP BRU IZC XWY ZIS
The string index received from P1:
10 11 12 13 14
The received index from P2 is 14
[kern@artixcse23] Problem_2_MessageQueue]$ _
```

Since the strings are generated randomly so the values of the strings might be different.

Compiling the Program

For compiling the program a makefile is supplied in the program directory. First, run make in a Linux terminal inside the program folder **Problem_2_MessageQueue.**
Then run the **main** file using **./main**.

Data Structures Used

- char[50][5] - The array for storing random string of length 3.
- int[5] - to store the index of the five strings to be sent.

- struct queueData - A custom structure used to store a long Index and a string of length 5 as mstring.
  struct queueData
  {
      long index;
      char mString[5];
  }
-

Logic of the program

- First we generate a character array of size char[50][5].
- Then we traverse through that array and fill the first three rows of each column with a random number generated using rand() which is between 65 and 100, the valid ASCII codes of A-Z.
- Then we select the 5 th index of the array , copy it to an another array toBeSent and an index array that contain the respective index.
- We send these two arrays to P2 using message queue and functions msgrcv and msgsnd to received and send messages respectively.
- P2 reads from those message queues using msgrcv, the strings and indexes required and sends the highest index back to the P1 using msgsnd.

Problem_2_Socket

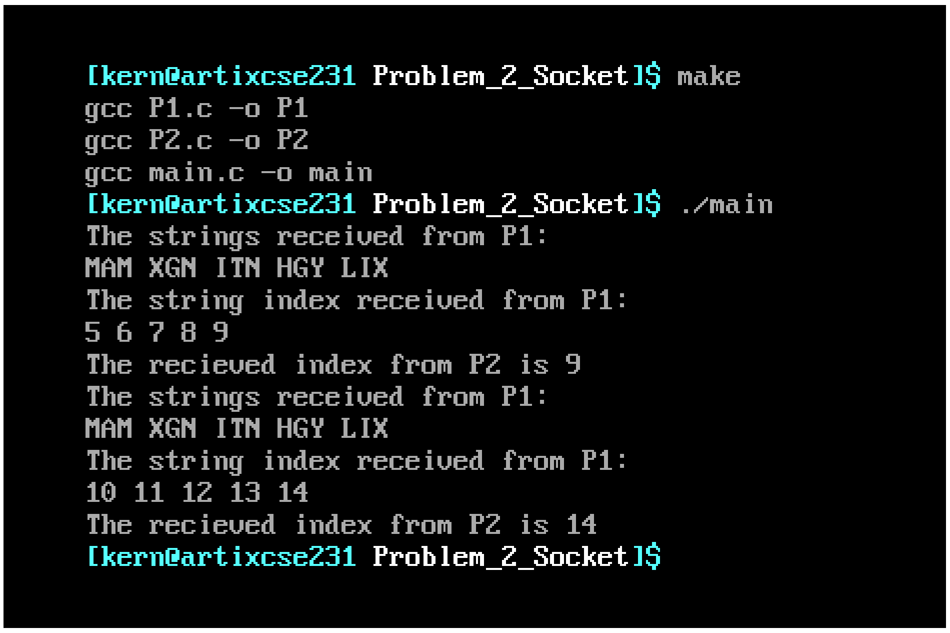
Input

No Input needs to be given. We start the program from index 5 of the random string array

Output

The first five string index starting from 5 would be printed on the string along with the randomly generated string content. Then the highest index would be sent back to P1.P1 now prints the received index 9. P1 now sends back the successor of that index 10, to the P2 which prints the indices and the content. P2 then sends the highest index 14 to the P1. P1 now prints the received index 14.

Expected Output



```
[kern@artixcse231 Problem_2_Socket]$ make
gcc P1.c -o P1
gcc P2.c -o P2
gcc main.c -o main
[kern@artixcse231 Problem_2_Socket]$ ./main
The strings received from P1:
MAM XGN ITN HGY LIX
The string index received from P1:
5 6 7 8 9
The recieved index from P2 is 9
The strings received from P1:
MAM XGN ITN HGY LIX
The string index received from P1:
10 11 12 13 14
The recieved index from P2 is 14
[kern@artixcse231 Problem_2_Socket]$
```

Since the strings are generated randomly so the values of the strings might be different.

Compiling the Program

For compiling the program a makefile is supplied in the program directory. First, run make in a Linux terminal inside the program folder **Problem_2_Socket.**

Then run the **main** file using **./main**.

Data Structures Used

- char[50][5] - The array for storing random string of length 3.
- int[5] - to store the index of the five strings to be sent.
- Unix Domain Sockets

Logic of the program

- First, we generate a character array of size char[50][5].
- Then we traverse through that array and fill the first three rows of each column with a random number generated using rand() which is between 65 and 100, the valid ASCII codes of A-Z.
- Then we select the 5 th index of the array, copy it to another array toBeSent and an index array that contain the respective index.
- We send these two arrays to P2 using sockets and write system call.
- P2 reads from those message queues using read system call, the strings and indexes required and sends the highest index back to the P1 using write system call.