PROJECT 3 :

1.**Rand() :**

Rand() Library call is checked in VOID IMAGE method of PIN program, It finds the routine by name "rand", if the routine is valid, it Inserts and calls the RandRtn method in the program, and in that method it checks for the mode Replay or Recording.
If the mode is recording, simply store the value of Rand() in trace.out with the prefix "**rand:**" for the replay mode.

When it is a replay mode, in the IMAGE method itself it checks for the ReplayMode and if it is, it change the routine Rand () by RTN_Replace(randRtn,AFUNPTR(MyRandFunc));.
MyRandFunc() is the function defined in my PIN program, which instead of calling Rand(), retrieves the recorded value of rand() and return it back to the program. The AFUNPTR is not needed to be instrumented and should have the same signature as that of Rand().
This is how record and replay of rand is done.

2. **Signals :**

Signals are something which are given at the runtime.
In the record mode, When the method Instruction is called, it also checks for the signal interruption. Since there are 16 signals, I have set the for loop from 1 to 16 which checks for if any signal has interrupted and the checking is done by calling **PIN_InterceptSignal (signal,INTERCEPTSIGNALCALLBACK,0);**

INTERCEPTSIGNALCALLBACK is the method which I have defined and it returns the bool value. Also in this method , I am tracing the signal into the file trace.out with the prefix "**signal:**"if there is any signal detected. This is how signals are recorded.

In the Replay Mode,
When my sysbefore is called, it starts reading from the file trace.out line by line. It calls for the method **CheckforSignal,** the method checks in the sequence if it is signal or not. If the prefix of the line matches with "signal :", it prints out the signal. And if the signal is "**^C**", then the program terminates as it terminates in record mode. So if the signal value is equal to 2, the program exits. If the prefix does not matches with "signal :", it continues with the system calls.

3. **System Calls**

Recording mode : In the recording mode, System before is called, which generates all the arguments, ctxt, nextAddr, and then SystemAfter is called which generates return value of that sys call and the error. In the SystemAfter, I am storing the name of the system call, arguments and ret value in the file trace.out with the prefix "**sys:**"

System Call:

For recording System call, I set the If condition in SysAfter for determining which System call.

1. If the system call Is read: it prints the value in buffer address of read and the ret value of the system call read in the trace.out.
2. If the system call is gettimeofday: it prints the values of the struct timeval in the file. That is, it prints

((time_t)sys_arg[0]) and *((time_t*)sys_arg[0]+1) in the file which stores the value of seconds and microseconds.

3. If the system call is open: it has a sys_arg[0] which stores the file name. Print this (char*)sys_arg[0] in the trace.out as a filename.
4. If the system call is sendto : then print the buffer value stored in sys_arg[1] in the file trace.out.
5. If the system call is accept: then print the value stored in the receive buffer values of struct sockadd whose address is in sys_arg[1], I.e print *((unsigned short*)sys_arg[1])
6. If the system call is rt_sigaction: then print the signum in the trace.out file.

Replaying System call:

Same as in recording, I am reading the file using fscanf when the condition of certain system call is true.This is done in SysBefore()

1. If the system call is read: then using fscanf read the value stored in file using %s and read the value of ret using %ld. Use the memcpy to copy this value to the current buffer location I.e at sys_arg[1].
2. If the system call is gettimeofday: then using fscanf, get the time in Seconds and microseconds and store in the …
3. If the system call is accept: then scan the sock address from the file and store it in the current sys_arg[0]
4. If the system call is receivefrom: then scan the buffer value and store it in the sendto buffer address.
5. If the system call is sendto: then again scan the buffer value and store it in the sendto buffer address.
6. If the system call is rt_sigaction: check if the signum==2 in the trace.out file and terminate.

Once the value is stored in destined location call the PIN_setcontext() to set the register value as return address of the syscall which was recorded and supposed to be replayed. Then to bypass the current system call, again call PIN_setcontext() and set the REG_INST_PTR with nextaddr of the instruction pointer. And the ExcecuteAt the current ctxt.