Exception-Less System Calls

L Chinmay

B M S College of Engineering

Abstract

For the last 30+ years, system calls have been used by the computers to service the request made by the user. Since then, Synchronous mechanism of execution of system calls has been accepted universally where a special processor instruction is used o yield user space execution to the kernel.

A new mechanism for handling the system calls has been proposed which is known as *exception-less system calls.* They enhance the efficiency of the processor and reduce the time required for the execution of the system calls. Exception-less system calls are particularly used in multicore processors where they target highly threaded server applications.

1 Introduction

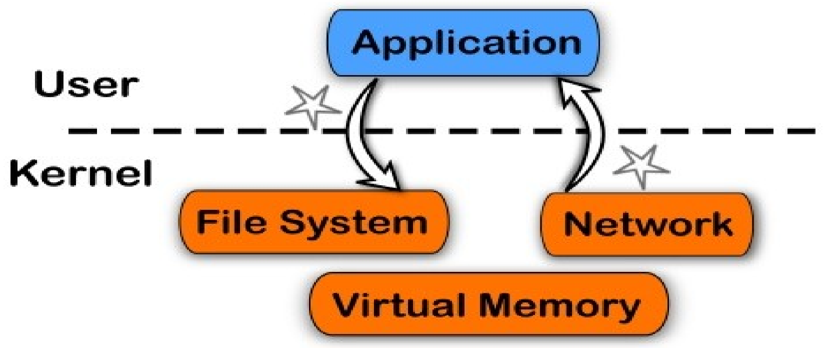
System calls hold a special position to service the request made by the user. Synchronous Mechanism of execution of system calls has been accepted universally. While different operating systems offer variety of services but the mechanism of execution of system calls is same in all these systems. This Synchronous mechanism typically involves writing the arguements into appropriate registers and issuing special machine instruction that raises an synchronous exception, immediately yielding user mode execution to kernel mode exception handler. Two important properties of this traditional method are:

(1). a processor exception is used to communicate with the kernel.

(2). Synchronous mechanism is enforced, as application expects the completion of the execution of system calls before resuming the user-mode execution.

Both of these effects result in performance inefficiencies on modern processors.

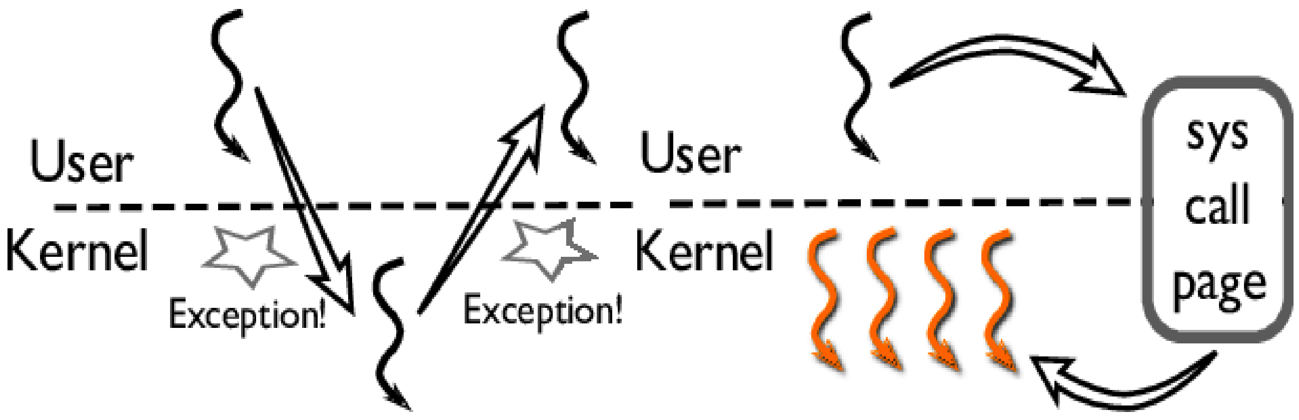
Synchronous system calls are expensive



Traditional system calls are synchronous and use exceptions to cross domains

7

To improve locality in the execution of system intensive workloads and to address (and partially eliminate) the performance impact of traditional, a new operating system mechanism can be implemented: the exception-less system call. An exception-lesssystem call is a mechanism for requesting kernel services that does not require the use of synchronous processor exceptions. In this implementation, system calls are issuedby writing kernel requests to a reservedsyscall page, us-ing normal memory store operations. The actual execution of system calls is performed asynchronously by special in-kernel syscall threads, which post the results of system calls to the syscall page after their completion. Decoupling the system call execution from its invocation creates the possibility for flexible system call scheduling, offering optimizations along two dimensions. The first optimization allows for the deferred batch execution of system calls resulting in increased temporal locality ofexecution. The second provides the ability to execute sys-tem calls on a separate core, in parallel to executing user-mode threads, resulting in spatial, per core locality. In both cases, system call threads become a simple, but powerful abstraction. The key benefit of exception-less system calls is the flexibility in scheduling system call execution, ultimately providing improved locality of execution of both user and kernel code.



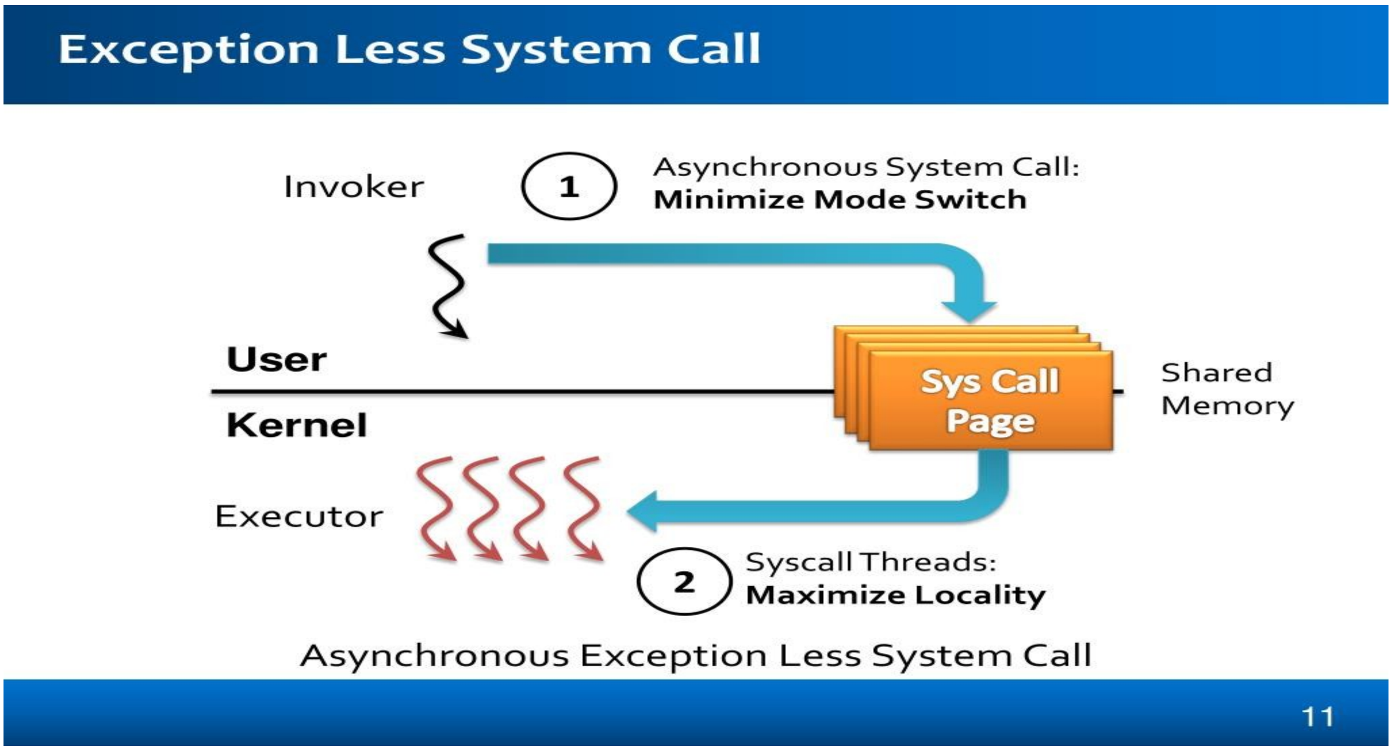
System call batching: Delaying the execution of a series of system calls and executing them in batches minimizes the frequency of switching between user and kernel execution, eliminating some of the mode switch overhead and allowing for improved temporal locality. This improves both the direct and indirect costs of system calls. Core specialization: In multicore systems, exception- less system calls allow a system call to be scheduled on a core different than the one on which the system call was invoked. Scheduling system calls on a separate processor core allows for improved spatial locality. The design of exception-less system calls consists of two components:

(1) an exception-less interface for user space threads to register system calls

(2) an in-kernel threading system that allows the delayed (asyn-chronous) execution of system calls, without interrupting or blocking the thread in userspace.

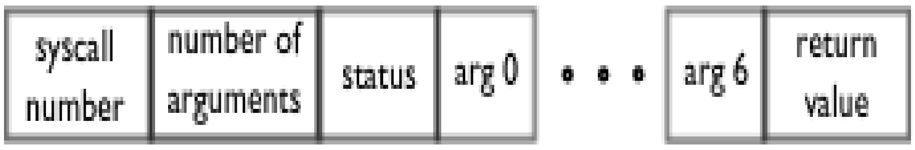
2 Exception-Less Syscall Interface

The interface for exception-less system calls is simply a set of memory pages that is shared amongst user and kernel space. The shared memory page, henceforth referred to as syscall page, is organized to contain exception-less system call entries. Each entry contains space for the request status, system call number, arguments, and return value.With traditional synchronous system calls, invocation occurs by populating predefined registers with system call information and issuing a specific machine instruction that immediately raises an exception.



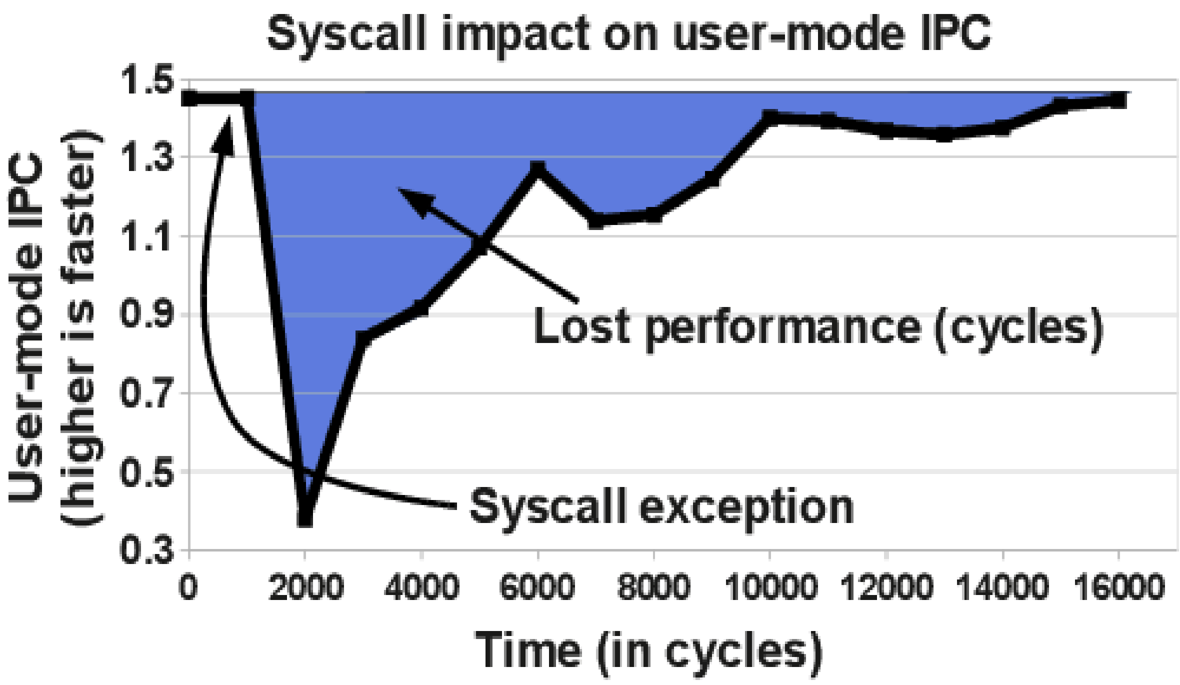
In contrast, to issue an exception-less system call, the user-space threads mustfind a free entry in the syscall page and populate the entry with the appropriate values using regular store instructions. The user-space thread can then continue executing without interruption. It is the responsibility of the user-space thread to later verify the completion of the system call by reading the status information in the entry. None of these operations, issuing a system call or verifying its completion, causes exceptions to be raised.

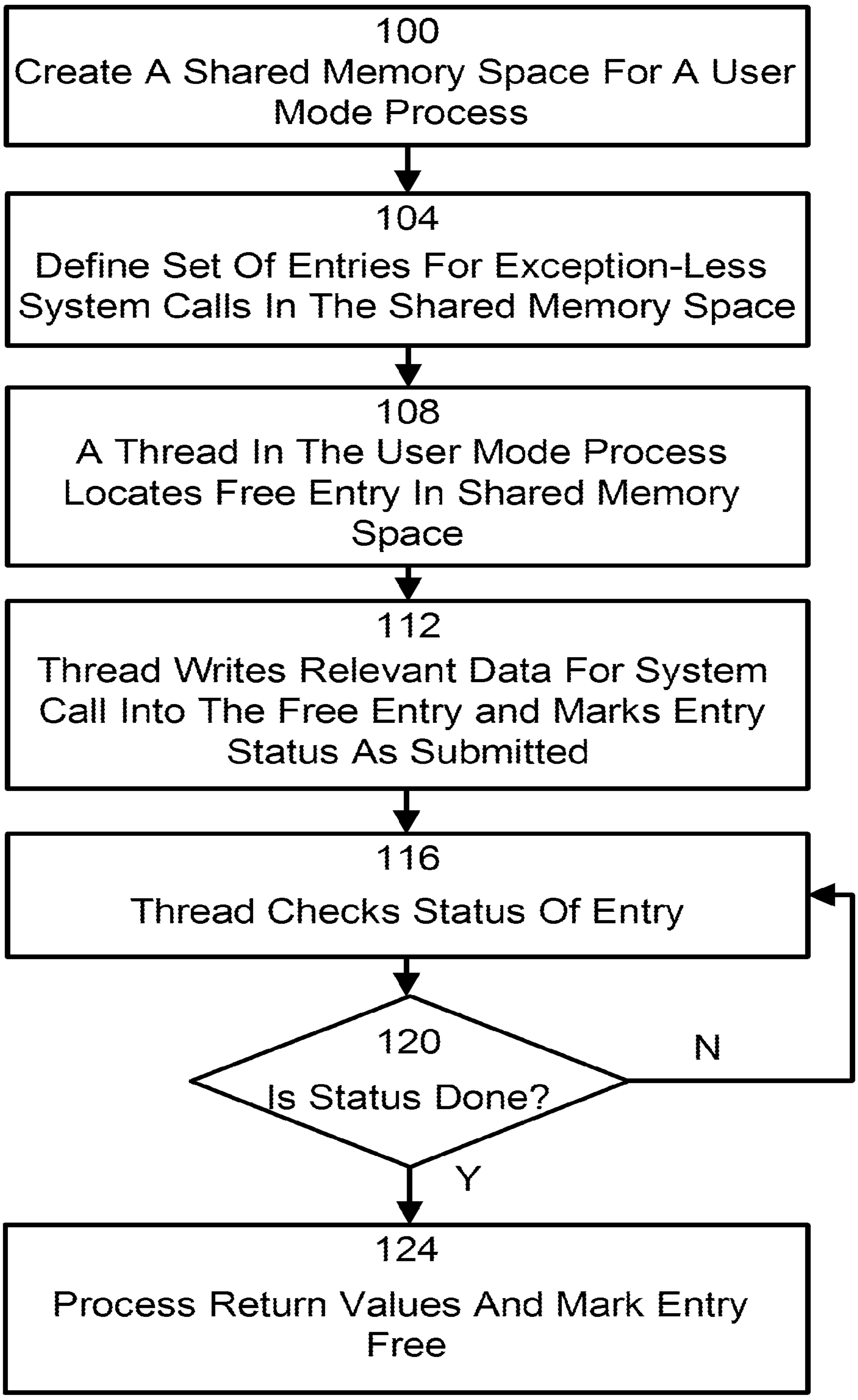
3 Syscall Pages



syscall number	number of arguments	status	arg 0	• • •	arg 6	return value

Syscall pages can be viewed as a table of syscall entries, each containing information specific to a single system call request, including the system call number, arguments, status (free/submitted/busy/done), and the result. In our 64-bit implementation, we have organized each entry to occupy 64 bytes. This size comes from the Linux ABI which allows any system call to have up to 6 arguments, and a return value, totalling 56 bytes. Although the remaining 3 fields (syscall number, status and number of arguments) could be packed in less than the remaining 8 bytes, we selected 64 bytes because 64 is a divisor of popular cache line sizes of today's processor.To issue an exception-less system call, the user-space thread must find an entry in one of its syscall pages that contain a free status field. It then writes the syscall number and arguments to the entry. Lastly, the status field is changed to submitted 4 , indicating to the kernel that the request is ready for execution. The thread must then check the status of the entry until it becomes done, consume the return value, and finally set the status of the entry to free.




100
Create A Shared Memory Space For A User Mode Process
104
Define Set Of Entries For Exception-Less System Calls In The Shared Memory Space
108
A Thread In The User Mode Process Locates Free Entry In Shared Memory Space
112
Thread Writes Relevant Data For System Call Into The Free Entry and Marks Entry Status As Submitted
116
Thread Checks Status Of Entry
120
Is Status Done?
N
Y
124
Process Return Values And Mark Entry Free

Implementation

```
import os
a=[["NULL" for i in range(10)] for j in range(20)]
b="available"
c="not available"
dict={
            "write": [100,3],
            "read": [101,3],
            "open": [102,3],
            "close": [103,1],
            "terminate": [104,1],
            "end": [105,1],
            "abort": [106,1],
            "fork": [107,0],
            "wait": [108,1],
            "exit": [109,1],
            "sleep": [110,1],
            "alarm": [111,2],
            "getppid": [112,0],
            "pipe": [113,1],
            "send": [114,2],
            "receive": [115,2],
            "cancel": [116,3],
            "testcancel": [117,2],
            "get_process_attribute":[118,4],
           "set_process_attribute": [119,4],
           "exec": [120,4]
}
t={
            "write": [100,3],
            "read": [101,3],
            "open": [102,3],
            "close": [103,1],
            "terminate": [104,1],
            "end": [105,1],
            "abort": [106,1],
            "fork": [107,0],
            "wait": [108,1],
            "exit": [109,1],
            "sleep": [110,1],
            "alarm": [111,2],
            "getppid": [112,0],
            "pipe": [113,1],
            "send": [114,2],
            "receive": [115,2],
            "cancel": [116,3],
            "testcancel": [117,2],
            "get_process_attribute":[118,4],
           "set_process_attribute": [119,4],
           "exec": [120,4]
}
y={}
for key in dict.keys():
            y[key]=0
a[0]=dict["read"]
a[0].extend(["not available","unsigned int fd","char *buf","size_t
count","NULL","NULL","NULL",1])
y["read"]=1
a[1]=dict["open"]
a[1].extend(["not available","unsigned int fd","const char *buf","size_t
count","NULL","NULL","NULL",1])
y["open"]=1
a[2]=dict["wait"]
a[2].extend(["not available","int time","NULL","NULL","NULL","NULL","NULL",0])
y["wait"]=1
a[3]=dict["send"]
a[3].extend(["not available","char *msg","pid id","NULL","NULL","NULL","NULL",1])
y["send"]=1
a[4]=dict["receive"]
a[4].extend(["not available","char *msg","pid id","NULL","NULL","NULL","NULL",1])
y["receive"]=1
a[5]=dict["get_process_attribute"]
a[5].extend(["not available","char *per","char *mode","pid id","char *type",
"NULL","NULL",1])
y["get_process_attribute"]=1
for i in range(6,20):
            a[i][2]=b

def isc(e):
            if((e>='a' and e<='z') or(e>='A' and e<='Z')):
                    return 1
            elif(e==''):
                    return 0
l=6
def exec(s):
            global l
            i=l
            for key in dict.keys():
                    if(s==key):
                            a[i][0]=dict[key][0]
                            a[i][1]=dict[key][1]
                            a[i][2]=c
                            d=t[s][1]
                            k=3
                            y[s]+=1
                            a[i][1]=d
                            a[i][0]=t[key][0]
                            if(s=="read" or s=="write" or s=="open" or s=="close"):
                                    w=input("Enter the File: ")
                                    while (not os.path.isfile(w)) or (not os.path.exists(w)):
                                            w= input("Whhoops! No such file! Please enter the
name of the file you'd like to use: ")
                                    if(d>1):
                                            print("Enter the arguements: ")
                                            for j in range(d-1):
                                                    z=str(input())
                                                    a[i][k]=z
                                                    k+=1
                            else:
                                    if(d>0):
                                            print("Enter the arguements: ")
                                            for j in range(d):
                                                    z=str(input())
                                                    a[i][k]=z
                                                    k+=1
                            for j in range(k,9):
                                    a[i][j]="NULL"
                            x=str(input("enter the return value: "))
                            a[i][9]=x
                            i+=1
                            print(s+" is added to the table ...!")
                            for m in range(20):
                                    print(a[m],end="\n \n")
            if(i==20 and a[0][0]=="NULL"):
                    l=0
            elif(i==20 and a[0][0]!="NULL"):
                    comp()
            elif(a[i][0]!="NULL"):
                    l=0
            else:
                    l=i
```

```
def comp():
        global l
        j=0
        if(j<20):
                while(j<20 and a[j][2]=="available" ):
                        j+=1
                if( j< 20 and a[j][2]=="not available"):
                        for key in dict.keys():
                                if(a[j][0]==t[key][0] and y[key]!=0):
                                        q=key
                                        print("after the completion of the execution of the
"+q+" system call in the table: ",end="\n")
                                        for k in range(2):
                                                a[j][k]="NULL"
                                        for k in range(3,len(a[j])):
                                                a[j][k]="NULL"
                                        a[j][2]=b
                                        y[key]-=1
                                        for m in range(20):
                                                print(a[m],end="\n \n")
                                        print(q+" has completed its execution ",end="\n")

                        if(a[0][0]=="NULL" and a[j+1][0]=="NULL"):
                                l=0
                elif(j>=20):
                        return

while(1):
        q=str(input())
        if(isc(q)==1):
                s=str(input("enter the command: "))
                if s in dict.keys():
                        exec(s)
                else:
                        print("Invalid System Call...")
        elif(isc(q)==0):
                e=1
                for i in range(len(a)):
                        if(a[i][0]!="NULL"):
                                e=0
                                break
                if(e==0):
                        print("During the execution of the system calls in the table: ")
                        comp()
                else:
```

```
print("All System Calls in the table are executed...")
l=0
i=0
```

References

01. Free Software Foundation. The GNU C ibrary.

02. M.M. Lehman. Programs, life cycles, and laws of software evolution. Proceedings of the IEEE, 68(9):1060–1076, Sept 1980. Michael Kerrisk.

03. Mike Hayward. LKML: Mike Hayward: Intel P6 vs P7 system call performance. https://lkml.org/lkml/2002/12/9/13

04. Intel Corporation. Intel 64 and IA-32 Architectures Software Developer’s Manual Volume 2: Instruction Set Reference, A-Z. https://goo.gl/1cFOvB

05.Linux programmer’s manual: Linux system calls

http://www.man7.org/linux/man-pages/man2/syscalls

06.Livio Soares, Michael Stumm FlexSC: Flexible System Call Scheduling with Exception-Less System Calls