

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ciencias de la Computación e Informática

CI-0122 Sistemas Operativos
Grupo 01
I Semestre

II Tarea programada: NachOS, System calls

Profesor:
Francisco Arroyo

Estudiantes:
Leonardo Barrientos Cerdas | B50946

14 de Junio 2019

Índice

1. Introducción	3
2. Objetivos	4
3. Descripción	5
4. Diseño	6
5. Desarrollo	7
6. Manual de usuario	19
Requerimientos de Software	19
Compilación	19
Especificación de las funciones del programa	19
7. Casos de Prueba	20

1. Introducción

La segunda tarea de nachOS consiste en implementar el manejo de excepciones y llamados al sistema. Debe ser capaz de soportar todos los llamados ya definidos, los cuales son:

- Halt, Exit, Exec, Join, Create, Open, Write, Read, Close, Fork, Yield, SemCreate, SemDestroy, SemSignal, SemWait.

Para lograr el objetivo es necesario el uso de "bitMap.h" para controlar la memoria física y se puedan colocar varios programas de usuario en la memoria principal a la vez.

2. Objetivos

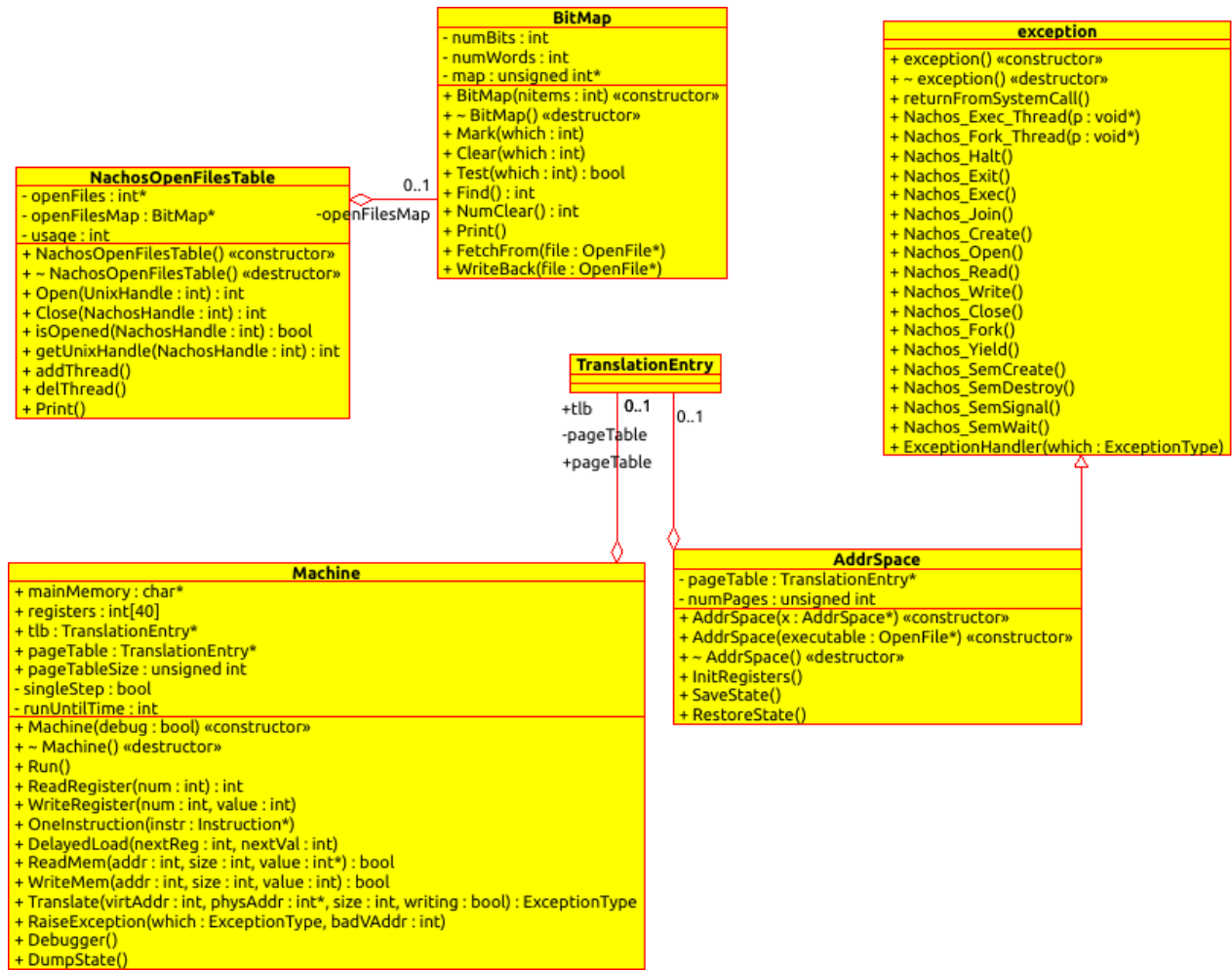
[Indicar los objetivos específicos para cada tarea, están en `os.ecci.ucr.ac.cr/ci1310`]

- Implantar el manejo de excepciones y llamados al sistema. Se deben soportar todos los llamados al sistema definidos en "syscall.h". Presentamos una rutina en ensamblador "syscall" que provee la manera de invocar un llamado al sistema desde una rutina C.
- Implantar multiprogramación. El código que presentamos le restringe a solo poder correr un programa de usuario a la vez.
- Implantar programas de usuario multi-hilos. Implante los llamados al sistema "forkz zield", que le permita al usuario llamar a una rutina en el mismo espacio de direccionamiento, y hacer ping pong entre los hilos

3. Descripción

- La segunda fase de NachOS es dar soporte a la multiprogramación. De la misma forma que en la primera asignación le damos parte del código que necesita; su trabajo será completar el sistema y mejorarlo. Hasta ahora el código que ha escrito para NachOS ha sido solo parte del kernel del sistema operativo. En un sistema operativo real, el kernel no solo utiliza sus procedimientos internamente, sino que también permite a programas de los usuarios tener acceso a algunas de sus rutinas utilizando llamados al sistema (system calls)
- El primer paso para desarrollar esta asignación es leer y entender la parte del sistema que hemos escrito para ustedes. Los archivos del kernel se encuentran en `userprog`, se adicionan algunas partes para la simulación de la máquina Mips en el directorio `machine` un sistema de archivos inicial en `filesys`. Los programas de los usuarios están en `testz` los utilitarios para crear los programas ejecutables de NachOS están en `bin`. Tenga presente que los programas de usuarios deben ser compilados y correr en máquinas con arquitectura Mips (con las que no cuenta la ECCL, por lo que es necesario utilizar un cross-compiler). NachOS debe ser compilado utilizando los utilitarios de las máquinas Linux
- Con el código que le presentamos, NachOS solo puede correr un programa de usuario codificado en C a la vez. Como caso de prueba, presentamos un programa de usuario trivial, `"halt"`, que lo que hace es utilizar un llamado al sistema para hacer que el sistema operativo 'apague' la máquina. Para correr el programa `"halt"`, usted debe compilar (`make`) y correr NachOS (`./nachos -x ../test/halt`). Debe seguir la ejecución, como se comporta cuando el programa es cargado en memoria, corre y luego invoca un llamado al sistema (`halt`)
- En esta asignación les estamos presentando una CPU simulada que modela una CPU real. De hecho, la CPU simulada es la misma que la CPU real (un chip Mips), pero simulando la ejecución, se tiene el control completo de cuántas instrucciones se ejecutan, cómo trabaja el espacio de direcciones, y cómo se manipulan las excepciones (incluyendo los llamados al sistema) e interrupciones
- Nuestro simulador puede correr programas escritos en C y compilados para la arquitectura Mips, ver el directorio `test` para encontrar algunos ejemplos. Los programas compilados (utilizando un cross-compiler) deben ser enlazados con una serie de banderas especiales (ver el `"Makefile"`), luego convertidos al formato de NachOS utilizando el utilitario `coff2noff` (que se provee en `bin`). La única restricción es que no se soportan las instrucciones de punto flotante del Mips
- Usted debe proveer un kernel de NachOS que sea a 'prueba de balas' contra los programas del usuario, no debe haber nada dentro de un programa del usuario que haga que el sistema operativo se 'caiga' (con la excepción de explícitamente llamar a la función `"halt"`)

4. Diseño



5. Desarrollo

[Acá se debe indicar cómo resolvieron el problema.]

Primeramente para lograr esta tarea fue necesario crear una clase de tabla de archivos abiertos, la cual maneja los archivos basado en unixHandle y utilizando el bitMap, para determinar si hay un archivo abierto, obtener el handle de unix de un archivo entre otras.

Se puede ver el .h a continuación:

```
1
2 #include "nachosTable.h"
3 #include <iostream>
4
5 using namespace std;
6
7 // se crea vector de openFile, el bitMap se inicializa
8 NachosOpenFilesTable::NachosOpenFilesTable()
9 {
10     openFiles = new int[PAGES];
11     openFilesMap = new BitMap(PAGES);
12
13     for (int i = 0; i < 3; i++) // marcar stdin, stdout y stderr
14     {
15         openFiles[i] = i;
16         openFilesMap->Mark(i);
17     }
18
19     usage = 0;
20 }
21
22 NachosOpenFilesTable::~NachosOpenFilesTable()
23 {
24     delete openFiles;
25     delete openFilesMap;
26 }
27
28 int NachosOpenFilesTable::Open( int UnixHandle )
29 {
30     int free = openFilesMap->Find(); // Devuelve el primer bit libre, y lo marca
31                                     // como en uso (busca y lo aloca)
32     if (free != -1)
33     {
34         openFiles[free] = UnixHandle; //asigna el unixHandle al archivo libre
35     }
36     return free;
37 }
38
39 // si se cierra devuelve 0 sino -1
40 int NachosOpenFilesTable::Close( int NachosHandle )
41 {
42     int close = -1;
43     if (isOpen(NachosHandle))
44     {
45         openFilesMap->Clear(NachosHandle); // Limpia el NachosHandle'esimo numero
```

```

45     openFiles[NachosHandle] = 0;
46     close = 0;
47 }
48 return close;
49 }
50
51 bool NachosOpenFilesTable::isOpenned( int NachosHandle )
52 {
53     return openFilesMap->Test(NachosHandle);    // retorna true si NachosHandle esta
54         seteado
55 }
56
57 // devuelve el identificador si esta abierto(si lo esta usando), sino devuelve -1
58 int NachosOpenFilesTable::getUnixHandle( int NachosHandle )
59 {
60     // printf("NachosHandle ----> %d", NachosHandle);
61     int identifier = -1;
62     if (isOpenned(NachosHandle))
63     {
64         identifier = openFiles[NachosHandle];
65     }
66     return identifier;
67 }
68
69 // aumenta en uno la cantidad de threads
70 void NachosOpenFilesTable::addThread()
71 {
72     usage++;
73 }
74
75 // decrementa en uno la cantidad de threads
76 void NachosOpenFilesTable::delThread()
77 {
78     usage--;
79 }
80
81 // Imprime ambos handle de cada openFile.
82 void NachosOpenFilesTable::Print()
83 {
84     for (int i = 0; i < PAGES; i++)
85     {
86         cout << "Nachos: " << i << "\n" << "Unix: " << openFiles[i] << endl;
87     }
88 }

```


Una vez creada la tabla, es necesario la modificación de la clase "AddrSpace", ya que es necesario para la implementación de algunos systemCalls y también es necesario agregar una estructura de tipo de bitMap a la clase "system" para poder representar las páginas de la memoria.

```

1
2 #ifndef USER_PROGRAM
3 #include "machine.h"
4 extern Machine* machine;          // user program memory and registers
5
6 extern BitMap* MiMapa;           // Declares a global variable defined elsewhere
7 #endif

```

En cuanto al addSpace fue necesario crear un nuevo constructor en el cual se asignan los datos del parámetro recibido a pageTable para las páginas, además se agregan las páginas de pila y se buscan páginas físicas. También el destructor lo que hace sería limpiar la estructura creada de bitMap (MiMapa).

```

1
2 AddrSpace::AddrSpace(AddrSpace* x)
3 {
4     numPages = x->numPages;
5     pageTable = new TranslationEntry[numPages];
6
7     int paginas = numPages - 8;
8     for(int i = 0; i < paginas; i++)
9     {
10         pageTable[i].virtualPage = x->pageTable[i].virtualPage;
11         pageTable[i].physicalPage = x->pageTable[i].physicalPage;
12         pageTable[i].valid = x->pageTable[i].valid;
13         pageTable[i].use = x->pageTable[i].use;
14         pageTable[i].dirty = x->pageTable[i].dirty;
15         pageTable[i].readOnly = x->pageTable[i].readOnly;
16     }
17
18     for (int i = paginas; i < numPages; i++)
19     {
20         pageTable[i].virtualPage = i;
21         pageTable[i].physicalPage = MiMapa->Find();
22         pageTable[i].valid = true;
23         pageTable[i].use = false;
24         pageTable[i].dirty = false;
25         pageTable[i].readOnly = false;
26     }
27 }
28
29 AddrSpace::~~AddrSpace()
30 {
31     for(int i = 0; i < numPages; i++)
32     {
33         MiMapa->Clear(pageTable[i].physicalPage); //Se limpian todas las paginas del
34         bitmap
35     }
36 }

```

También se agregó la implementación del método saveState() y se completó el de restoreState(). En el caso del safe si la memoria virtual está definida se recorre el pageTable para poner el use y dirty de las

páginas, luego se crea un nuevo translation lookaside buffer y ponemos las paginas en false, esto para el pageFaultException. Por otro lado el restore se le agrega para restaurar el estado.

```
1
2
3 void AddrSpace::SaveState() {
4     #ifdef VM //Si la memoria virtual esta definida
5
6         for(int i = 0; i < TLBSize; ++i) //poner el uso y el
            dirty de las paginas igual a como este en el translation lookaside buffer
7         {
8             pageTable[machine->tlb[i].virtualPage].use = machine->tlb[i].use;
9             pageTable[machine->tlb[i].virtualPage].dirty = machine->tlb[i].dirty;
10        }
11
12        machine->tlb = new TranslationEntry[TLBSize]; //nuevo translation lookaside
            buffer
13        for (int j = 0; j < TLBSize; ++i)
14        {
15            machine->tlb[j].valid = false; //para poder hacer
            pageFaultException
16        }
17    #endif
18 }
19
20
21 void AddrSpace::RestoreState()
22 {
23
24    #ifndef VM
25        machine->pageTable = pageTable;
26        machine->pageTableSize = numPages;
27    #else //Se restaura el estado
28        machine->tlb = new TranslationEntry[TLBSize];
29        for (int i = 0; i < TLBSize; ++i)
30        {
31            machine->tlb[i].valid = false;
32        }
33    #endif
34 }
```

Por último se tuvieron que programar todos los syscalls que estaban definidos en "syscall.h", estos se tuvieron que agregar ala clase .exception.c", y tener un switch para manejar la excepción.

```
1
2 // System call # 0
3 void Nachos_Halt() {
4
5     DEBUG('a', "Shutdown, initiated by user program.\n");
6     interrupt->Halt();
7
8 } // Nachos_Halt
9
10
11 // System call # 1
```

```

12 void Nachos_Exit()
13 {
14     int status = machine->ReadRegister(4);
15     IntStatus oldLevel = interrupt->SetLevel(IntOff);
16     // Para pruebas
17     // if(status == 0){
18     //     printf("Proceso Termino con %d \n", status);
19     // }
20
21     //Busca en el mapa conteniendo estructuras de semaforos y la cantidad de ellos
22     // si hay mas de un hilo o bien si este no es el ultimo
23     if(threadMap.find((long)currentThread) != threadMap.end()){
24         printf("Corriendo joins.\n");
25         //recorre la estructura haciendole signal a todos los semaforos que se encuentren
26         // en wait
27         threadStruct* d = threadMap[(long)currentThread];
28         for(int i = 0; i < d->threadNum; ++i){
29             d->structSem->V();
30         }
31     }
32     //Borra el hilo
33     threadMap.erase((long)currentThread);
34
35     //Recorre los hilos corriendo el siguiente si hay o bien terminandolos si ya no
36     // hay mas
37     Thread *nextThread;
38     nextThread = scheduler->FindNextToRun();
39
40     if(nextThread != NULL){
41         // printf("Cambiano al proximo hilo %s.\n",nextThread->getName());
42         scheduler->Run(nextThread);
43     }else{
44         // printf("Terminando hilo: %s.\n",currentThread->getName());
45         currentThread->Finish();
46     }
47     interrupt->SetLevel(oldLevel);
48
49     returnFromSystemCall();
50 }
51
52 // System call # 2
53 void Nachos_Exec()
54 {
55     DEBUG('n', "Start Exec.\n");
56     // Crea un hilo y una estructura para guardar en el mapa
57     Thread *hilito = new Thread("new thread");
58     threadStruct* data = new threadStruct;
59     data->threadNum = 0;
60     data->structSem = new Semaphore("Thread Semaphore", 0);
61     threadMap[(long)hilito] = data;
62
63     //Ejecuta el open de la tabla de archivos basado en el id del hilo actual
64     int spaceId = currentThread->open_files->Open((long)hilito);
65
66     (spaceId == -1) ? printf("Fail.\n") : printf("Thread Save\n");

```

```

65     machine->WriteRegister( 2, spaceId );
66
67     //Ejecuta el nuevo hilo manda como parametro la direccion del exec donde esta
68     //el nombre del archivo (parametro)
69     hilito->Fork(Nachos_Exec_Thread, (void*)machine->ReadRegister(4));
70
71     returnFromSystemCall();
72     DEBUG('n', "End Exec.\n");
73 }
74
75
76 // System call # 3
77 void Nachos_Join()
78 {
79     SpaceId id = machine->ReadRegister(4); //Se lee el id del hilo
80
81     // Limpiamos el bitmap de archivos ejecutables y hacemos wait, luego escribimos 0
82     // en registro 2, sino -1
83     if(execFilesMap->Test(id))
84     {
85         Semaphore* semaforo = new Semaphore("Join_sema", 0);
86         exFile[id]->sema = semaforo;
87         semaforo->P(); //wait
88         execFilesMap->Clear(id);
89         machine->WriteRegister(2,0);
90         delete exFile[id]; // borro el id del vec de
91         // punteros
92     }else
93     {
94         machine->WriteRegister(2,-1);
95     }
96     returnFromSystemCall();
97 }
98
99 // System call # 4
100 void Nachos_Create()
101 {
102     //Lee el nombre del archivo y lo crea con el creat de unix usando este nombre
103     int register4 = machine->ReadRegister(4);
104     char name[128] = "";
105
106     for (int i = 0, c = 1; c != 0; i++)
107     {
108         machine->ReadMem(register4, 1, &c);
109         name[i] = c;
110         register4++;
111     }
112
113     int id = creat((const char*)name, O_CREAT | S_IRWXU);
114     printf("Creado correctamente archivo %d....\n", id);
115     close(id);
116     machine->WriteRegister(2, 0);
117     returnFromSystemCall();

```

```

118 }
119
120
121 // System call # 5
122 void Nachos_Open()
123 {
124     int register4 = machine->ReadRegister(4);
125     int lect = 0;
126     int index = 0;
127
128     char name[FILE_SIZE] = {0};
129     machine->ReadMem(register4, 1, &lect); // lee el registro 4 y lo guarda en lect. (
        el 1 significa un byte)
130
131     // mientras lect no est vac o, se coloca su coontenido en el vector name.
132     while (lect != 0)
133     {
134         name[index] = lect;
135         register4++;
136         index++;
137         machine->ReadMem(register4, 1, &lect);
138     }
139
140     /*-----
141
142     int unix_file = open(name, O_RDWR); // abrir archivo de unix | O_RDWR= read &
        write
143
144     /*
145         si se abre correctamente, se busca espacio en tabla y se escribe en el reg 2 la
            pocisin,
146         si no se imprime un error y el nombre de archivo
147     */
148     if (unix_file != -1)
149     {
150         int nachos_file = currentThread->open_files->Open(unix_file);
151         if (nachos_file != -1)
152         {
153             machine->WriteRegister(2, nachos_file);
154             printf("Se abri correctamente %s...\n", name);
155         }
156     }
157     else
158     {
159         printf("ERROR.... %s no se pudo abrir ", name);
160     }
161
162
163     returnFromSystemCall(); // Update the PC registers
164
165 } // Nachos_Open
166
167
168 // System call # 6
169 void Nachos_Read()

```

```

170 {
171     // Se leen los reg 4,5,6 para ver direccion, tamano y id del archivo
172     int register4 = machine->ReadRegister(4);
173     int size = machine->ReadRegister(5);
174     OpenFileId id = machine->ReadRegister(6);
175
176     int totalChar = 0; //Guarda
177     cantidad de chars leidos
178     char buffer[size + 1] = {0}; // buffer que
179     guarda lo que lee
180     int last_i = 0;
181
182     switch(id)
183     {
184         case ConsoleInput: // Este es para el caso de la lectura por medio del teclado
185
186             printf("Estamos en ConsoleInput, tiene que terminar con (.): ");
187             for(int i = 0; i < size; i++, last_i = i)
188             {
189                 cin >> buffer[i]; //Se guarda en buffer[i] lo
190                 escrito por el usuario
191                 // printf("buffer[%d] = %d\n", i, buffer[i]);
192                 if(buffer[i] == '.')
193                     i = size;
194             }
195
196             buffer[last_i + 1] = '\0';
197             totalChar = strlen(buffer); //strlen: cantidad de chars
198             ingresados por el usuario
199
200             for(int index = 0; index < totalChar; index++)
201             {
202                 machine->WriteMem(register4,1,buffer[index]); //
203                 Almacenamiento en r4 lo escrito por el usuario
204                 register4 += 1;
205             }
206             machine->WriteRegister(2, totalChar); //Almacenando en el
207             registro 2 el total_caracteres
208
209             break;
210
211             case ConsoleOutput:
212                 printf("ERROR salida estandar!!!!\n");
213                 break;
214
215             case ConsoleError:
216                 printf("ERROR del error estandar!!!!\n");
217                 break;
218
219             default:
220                 /*
221                 Si el archivo est abierto en la TAA, trata de abrir el archivo, lo
222                 escribe en registro 4
223                 y en el 2 el numero de chars que se leyeron. De lo contrario escribe en
224                 registro 2; -1 nada m s.
225                 */
226                 if(currentThread->open_files->isOpened(id))

```

```

218         {
219             int open_unix_file_id = currentThread->open_files->
                getUnixHandle(id);
220             totalChar = read(open_unix_file_id, buffer, size);
221             for(int index = 0; index < totalChar; index++)
222             {
223                 machine->WriteMem(register4, 1, buffer[index]);
224                 register4 += 1;
225             }
226             machine->WriteRegister(2, totalChar);
227             printf("Archivo ID %d le do...\n", id);
228         }
229         else
230         {
231             machine->WriteRegister(2, -1); //Se escribe -1 en el
                registro 2
232             printf("ERROR leyendo archivo ID %d", id);
233         }
234         break;
235     }
236     returnFromSystemCall();
237 }
238
239 // System call # 7
240 void Nachos_Write()
241 {
242     // char * buffer = NULL;
243     int register4 = machine->ReadRegister(4);
244     int size = machine->ReadRegister(5); // Read size to write
245     OpenFileId id = machine->ReadRegister( 6 ); // Read file descriptor
246     char buffer[size+1] = {NULL};
247
248     int index;
249     int charsDone;
250     int letra;
251
252     machine->ReadMem(register4, 1, &letra);
253
254     // mientras no se sobrepase el tamaño, se va guardando en el buffer
255     while (index != size)
256     {
257         buffer[index] = letra;
258         register4++;
259         index++;
260         machine->ReadMem(register4, 1, &letra);
261     }
262
263     // Need a semaphore to synchronize access to console
264     console->P();
265     switch (id) {
266         case ConsoleInput: // User could not write to standard input
267             machine->WriteRegister( 2, -1 );
268             break;
269         case ConsoleOutput:
270             buffer[ size ] = 0;

```

```

272         printf( "%s\n", buffer );
273     break;
274     case ConsoleError:        // This trick permits to write integers to
        console
275         printf( "ERROR: %d\n", machine->ReadRegister( 4 ) );
276         break;
277     default:                // All other opened files
        // Verify if the file is opened, if not return -1 in r2
278     if (!currentThread->open_files->isOpen(id))
279     {
280         machine->WriteRegister(2, -1);
281     }else
282     {
283         // Get the unix handle from our table for open files
284         int unixId = currentThread->open_files->getUnixHandle(id);
285         // Do the write to the already opened Unix file
286         charsDone = write(unixId, buffer, size);
287         // Return the number of chars written to user, via r2
288         machine->WriteRegister(2, charsDone);
289     }
290
291     break;
292 }
293 // Update simulation stats, see details in Statistics class in machine/stats.cc
294 stats->numConsoleCharsWritten += size;
295 console->V();
296
297 returnFromSystemCall();        // Update the PC registers
298
299 }        // Nachos_Write
300
301
302 // System call # 8
303 void Nachos_Close(){
304     //Cierra el archivo basado en su id que esta guardado en la tabla de archivos
        abiertos TAA
305
306     // printf("estamos aqui");
307     OpenFileId id = machine->ReadRegister(4);        // leer el reg
        4
308     int unix_fileId = currentThread->open_files->getUnixHandle(id);        // identificador
        de la openFilesTable
309     int nachos_close = currentThread->open_files->Close(id);
310     int unix_close = close(unix_fileId);
311
312     //Si no cierra, imprimir ERROR
313     if ((nachos_close == -1 || unix_close == -1))
314     {
315         printf("ERROR cerrando archivo!!!!\n");
316     } else
317     {
318         printf("Se cerro correctamente....\n");
319     }
320
321
322     returnFromSystemCall();
323

```



```

324 }
325
326
327 // System call # 9
328 void Nachos_Fork()
329 {
330     DEBUG( 'u', "Entering Fork System call\n" );
331     // We need to create a new kernel thread to execute the user thread
332     Thread * threadsito = new Thread( "child to execute Fork code" );
333
334     // We need to share the Open File Table structure with this new child
335
336     threadsito->open_files = currentThread->open_files;           //Se asigna la
        tabla de archivos abiertos al nuevo hijo
337     threadsito->open_files->addThread();                           //Se aade el
        nuevo hilo
338
339     // Child and father will also share the same address space, except for the stack
340     // Text, init data and uninit data are shared, a new stack area must be created
341     // for the new child
342     // We suggest the use of a new constructor in AddrSpace class,
343     // This new constructor will copy the shared segments (space variable) from
        currentThread, passed
344     // as a parameter, and create a new stack for the new child
345     threadsito->space = new AddrSpace( currentThread->space );
346
347     // We (kernel)-Fork to a new method to execute the child code
348     // Pass the user routine address, now in register 4, as a parameter
349     // Note: in 64 bits register 4 need to be casted to (void *)
350     threadsito->Fork( Nachos_Fork_Thread, (void*)(long)machine->ReadRegister( 4 ) );
351     currentThread->Yield();
352     returnFromSystemCall();    // This adjust the PrevPC, PC, and NextPC registers
353
354     DEBUG( 'u', "Exiting Fork System call\n" );
355 }
356
357
358 // System call # 10
359 void Nachos_Yield()
360 {
361     // llamar al Yield de nachos, con current thread
362     currentThread->Yield();
363     returnFromSystemCall();
364 }
365
366
367 // // System call # 11
368 void Nachos_SemCreate()
369 {
370     int initVal = machine->ReadRegister(4);
371     Semaphore* sema = new Semaphore("semaforito", initVal);
372
373     /*
374     si se crea el semaforo correctamente, se busca un espacio en bitmap y se agrega
        el semaforo a la tabla
375     sino escribe -1 en el registro 2

```

```

376     */
377     if (sema != NULL)
378     {
379         int semId = sem_bitMap->Find();
380         semaphore_vec[semId] = sema;
381         currentThread->open_semaphores->Open(semId);
382         currentThread->open_semaphores->addThread();
383         machine->WriteRegister(2, semId);
384     }else
385     {
386         machine->WriteRegister(2, -1);
387     }
388
389     returnFromSystemCall();
390 }
391
392 // // System call # 12
393 void Nachos_SemDestroy()
394 {
395     int semaphore_id = machine->ReadRegister(4);
396
397     // si existe en tabla
398     if (sem_bitMap->Test(semaphore_id) == true)
399     {
400         currentThread->open_semaphores->Close(semaphore_id);
401         currentThread->open_semaphores->delThread();
402         sem_bitMap->Clear(semaphore_id);
403         machine->WriteRegister(2, 0);
404         delete semaphore_vec[semaphore_id];
405     }else
406     {
407         machine->WriteRegister(2, -1);
408     }
409
410     returnFromSystemCall();
411 }
412
413 // System call # 13
414 void Nachos_SemSignal()
415 {
416     int semaphore_id = machine->ReadRegister(4);    //Se lee el id del sem foro del
417     registro 4
418
419     if(sem_bitMap->Test(semaphore_id) == true)    //Si el sem foro existe en la
420     tabla
421     {
422         semaphore_vec[semaphore_id]->V();    //Se hace signal
423         machine->WriteRegister( 2, 0 );    //Se devuelve 0 en el registro
424         2
425     }else
426     {
427         machine->WriteRegister( 2, -1 );    //Se devuelve -1
428     }
429     returnFromSystemCall();
430 }

```

```

429 // System call # 14
430 void Nachos_SemWait()
431 {
432     int semaphore_id = machine->ReadRegister(4);
433
434     //si existe el semaforo en la tabla, se hace wait() y se escribe 0 en reg 2, si no
    //se escribe -1
435     if (sem_bitMap->Test(semaphore_id))
436     {
437         semaphore_vec[semaphore_id]->P();    // wait()
438         machine->WriteRegister(2, 0);
439     }else
440     {
441         machine->WriteRegister(2, -1);
442     }
443     returnFromSystemCall();
444 }

```

6. Manual de usuario

[Esta es una guía para indicar como utilizar el programa y qué se necesita.]

Requerimientos de Software

- **Sistema Operativo:** Linux
- **Arquitectura:** 32 bits o 64 bits
- **Ambiente:** consola

Compilación

Para compilar el programa, se utiliza `gcc` en la siguiente sentencia:

```
$ ./nachos -x ../test/[nombre del ejecutable]
```

Especificación de las funciones del programa

El programa solamente lee archivos que sean de la forma Nombre.c" y es necesario hacer make depend y make antes

7. Casos de Prueba

[En esta sección se espera encontrar las pruebas que se utilizaron para verificar la funcionalidad del programa elaborado. Pueden agregar pantallazos y/o partes de código. En algunas de las tareas, hay pruebas específicas que serán evaluadas (en su momento se indicarán), es deseable que se encuentren en este apartado.]

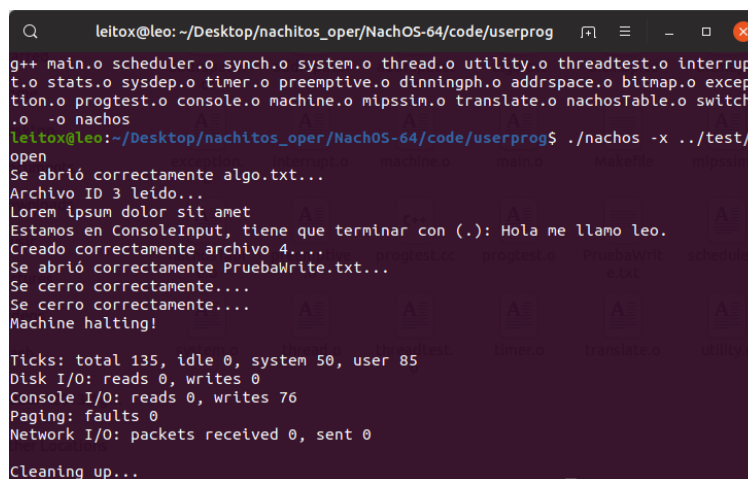
Prueba 1:

Esta prueba corresponde a la prueba del archivo open.c, esta prueba utiliza los syscalls Open, Read, Write y Close .

El código utilizado corresponde a:

```
1
2      #include "syscall.h"
3
4  int main(){
5      char arreglo[64];
6      char arreglo2[16];
7      int id = Open("algo.txt");
8      Read(arreglo, 64, id);
9      Write(arreglo, 64, 1);
10     Read(arreglo2, 64, 0);
11     Create("PruebaWrite.txt");
12     id = Open("PruebaWrite.txt");
13     Write(arreglo2, 12, id);
14     Close("algo.txt");
15     Close("PruebaWrite.txt");
16     Halt();
17     return 0;
18 }
```

Y el resultado obtenido se puede verificar en la figura 6:



```
Q leitox@leo: ~/Desktop/nachitos_oper/NachOS-64/code/userprog
g++ main.o scheduler.o synch.o system.o thread.o utility.o threadtest.o interrup
t.o stats.o sysdep.o timer.o preemptive.o dinningph.o addrspace.o bitmap.o excep
tion.o proptest.o console.o machine.o mipssim.o translate.o nachosTable.o switch
.o -o nachos
leitox@leo:~/Desktop/nachitos_oper/NachOS-64/code/userprog$ ./nachos -x ../test/
open
Se abrió correctamente algo.txt...
Archivo ID 3 leído...
Lorem ipsum dolor sit amet
Estamos en ConsoleInput, tiene que terminar con (.): Hola me llamo leo.
Creado correctamente archivo 4....
Se abrió correctamente PruebaWrite.txt...
Se cerro correctamente....
Se cerro correctamente....
Machine halting!

Ticks: total 135, idle 0, system 50, user 85
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 76
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

Figura 1: Salida del programa.

Prueba 2:

Esta prueba corresponde a la prueba del archivo agua.c, esta prueba utiliza los syscalls SemWait, SemSignal, SemCreate, SemDestroy.

El código utilizado corresponde a:

```
1
2 // agua.cc
3 // Test for user semaphore operations
4 //
5 //-----
6
7 int sH, sO;
8 int cH, cO;
9
10 void H() {
11
12     if(cH > 0 && cO > 0) {
13         cH--;
14         cO--;
15         Write("H haciendo agua!!\n", 17, 1);
16         SemSignal( sH );
17         SemSignal( sO );
18     }
19     else {
20         cH++;
21         Write("H esperando ...!!\n", 17, 1);
22         SemWait( sH );
23     }
24 }
25
26 void O() {
27
28     if( cH > 1) {
29         cH -= 2;
30         Write("O haciendo agua!!\n", 17, 1);
31         SemSignal( sH );
32         SemSignal( sH );
33     }
34     else {
35         cO++;
36         Write("O esperando ...!!\n", 17, 1);
37         SemWait( sO );
38     }
39 }
40
41 int main() {
42
43     int i;
44     cH = cO = 0;
45     sO = SemCreate( 0 ); // Creates the semaphores
46     sH = SemCreate( 0 );
47
48     for (i=0; i<30 ; i++) {
```

```

49         if ( (i % 2) == 0 )
50             Fork( H );
51         else
52             Fork( O );
53         Yield();
54     }
55
56     SemDestroy( sO );
57     SemDestroy( sH );
58
59     return 0;
60 }

```

Y el resultado obtenido se puede verificar en la figura 6:

```
H esperando ....!!
O esperando ....!!
H haciendo agua!!
O esperando ....!!
H esperando ....!!
O esperando ....!!
H haciendo agua!!
O esperando ....!!
H esperando ....!!
O esperando ....!!
H haciendo agua!!
O esperando ....!!
H esperando ....!!
O esperando ....!!
H esperando ....!!
O esperando ....!!
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 4306, idle 0, system 2090, user 2216
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 510
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Figura 2: Salida del programa.

Prueba 3:

Esta prueba corresponde a la prueba del archivo agua.c, esta prueba utiliza los syscalls SemWait, SemSignal, SemCreate, SemDestroy, Close y Create.

El código utilizado corresponde a:

```
1
2     int main(){
3         int fd;
4         int x = SemCreate(5);
5         Create("brbr.txt");
6         fd = Open("brbr.txt");
7         Write("wait", 4, fd);
8         SemWait(x);
9         SemSignal(x);
10        Write("signal", 6, fd);
11        Close(fd);
12
13
14        SemDestroy(x);
```

```

15     Halt ();
16 }

```

Y el resultado obtenido se puede verificar en la figura 6:

```

leitox@leo: ~/Desktop/nachitos_oper/NachOS-64/code/userprog
../userprog/exception.cc: In function 'void Nachos_Write()':
../userprog/exception.cc:413:32: warning: converting to non-pointer type 'char'
from NULL [-Wconversion-null]
    char buffer[size+1] = {NULL};
                               ^
g++ main.o scheduler.o synch.o system.o thread.o utility.o threadtest.o interrup
t.o stats.o sysdep.o timer.o preemptive.o dinningph.o addrspace.o bitmap.o excep
tion.o progtest.o console.o machine.o mipssim.o translate.o nachosTable.o switch
.o -o nachos
leitox@leo:~/Desktop/nachitos_oper/NachOS-64/code/userprog$ ./nachos -x ../test/
semaphore
Creado correctamente archivo 3....
Se abrió correctamente brbr.txt...
Se cerro correctamente....
Machine halting!

Ticks: total 152, idle 0, system 70, user 82
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 10
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...

```

Figura 3: Salida del programa.

Prueba 4:

Esta prueba corresponde a la prueba del archivo agua.c, esta prueba utiliza los syscalls Open, Close y Exit.

El código utilizado corresponde a:

```

1
2     #include "syscall.h"
3
4     main()
5     {
6         OpenFileId f1;
7
8
9         f1 = Open( "nachos.1" );
10        Close( f1 );
11
12        Exit( 0 );
13    }

```

Y el resultado obtenido se puede verificar en la figura 6:

Prueba 5:

Esta prueba corresponde a la prueba del archivo pingPong.c, esta prueba utiliza los syscalls Fork, Write y Yield.

El código utilizado corresponde a:

```

leitox@leo:~/Desktop/nachitos_oper/NachOS-64/code/userprog$ ./nachos -x ../test/
nachitos
Se abrió correctamente nachos.1...
Se cerró correctamente...
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 37, idle 0, system 10, user 27
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...

```

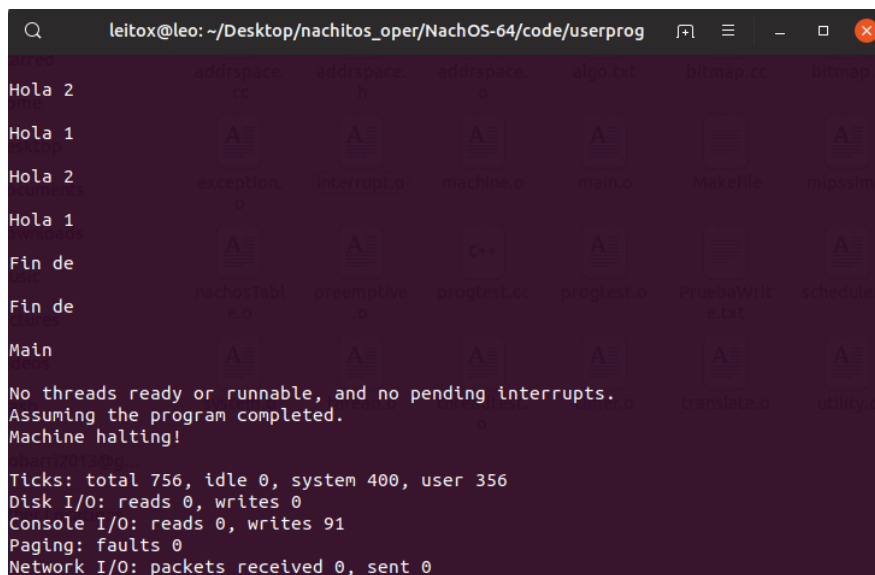
Figura 4: Salida del programa.

```

1
2     #include "syscall.h"
3
4     void SimpleThread(int);
5
6     int
7     main( int argc, char * argv[] ) {
8
9         Fork(SimpleThread);
10        SimpleThread(1);
11
12        Write("Main \n", 7, 1);
13        Write(argc, 4, 1);
14        Write(argv, 4, 1);
15    }
16
17
18    void SimpleThread(int num)
19    {
20
21        if (num == 1) {
22            for (num = 0; num < 5; num++) {
23                Write("Hola 1\n", 7, 1);
24                Yield();
25            }
26        }
27
28        else {
29            for (num = 0; num < 5; num++) {
30                Write("Hola 2\n", 7, 1);
31                Yield();
32            }
33        }
34        Write("Fin de\n", 7, 1);
35    }

```


Y el resultado obtenido se puede verificar en la figura 6:



```
leitox@leo: ~/Desktop/nachitos_oper/NachOS-64/code/userprog
Hola 2
Hola 1
Hola 2
Hola 1
Fin de
Fin de
Main
No threads ready or runnable, and no pending interrupts.
Assuming the program completed.
Machine halting!

Ticks: total 756, idle 0, system 400, user 356
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 91
Paging: faults 0
Network I/O: packets received 0, sent 0
```

Figura 5: Salida del programa.

Prueba 6:

Esta prueba corresponde a la prueba del archivo addSpacetest.c, esta prueba comprueba el funcionamiento de la clase addrSpace.

El código utilizado corresponde a:

```
1  #include "syscall.h"
2
3
4  /* Este proceso sirve para probar que el programa cargue correctamente las
5     p ginas  en el addrspace.
6     Requiere que se encuentre implementado el system call Write() y el system
7     call Exit() (aunque nicamente porque el programa lo llama al final)
8
9     Se recomienda que las p ginas f sicas en memoria se guarden en desorden (p.
10    e.
11    p gina virtual 1 en p gina f sica 2, p gina virtual 2 en p gina
12    f sica 4,
13    etc.)
14
15    El programa crear un buffer de 1024 bytes (4 p ginas ) y lo llena con
16    27 caracteres . Si el programa addrspace est correctamente implementado
17    debera escribir:
18
19    abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
20    abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
21    stuvwxyz{abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
22    abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
23    jklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
24    abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
25    abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
26    abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
27    abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
```

```

21     abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
stuvwxyz{abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
22     abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
23     abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
24     abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
abcdefghijklmnopqrstuvwxyz{abcdefghijklmnopqrstuvwxyz{
25     jklmnopqrstuvwxyz
26
27     */
28
29     void main () {
30         int i = 0, j = 0;
31         char buffer[1024];
32
33         for (j = 0; j<1024;j++) {
34             buffer[j]=(char)((j%27)+'a');
35         }
36
37
38         while (i<1) {
39             Write(buffer,1024,1);
40             i++;
41         }
42     }

```

Y el resultado obtenido se puede verificar en la figura 6:

[illegible]

Figura 6: Salida del programa.