Plancha 1

1. NachOS tiene 4096 bytes de memoria simulada.

```
const unsigned NUM_PHYS_PAGES = 32;
const unsigned SECTOR_SIZE = 128;
const unsigned PAGE_SIZE = SECTOR_SIZE;
const unsigned MEMORY_SIZE = NUM_PHYS_PAGES * PAGE_SIZE;
```

2. Este valor es dependiente de NUM_PHYS_PAGES y SECTOR_SIZE, así que se puede modificar mediante dichas variables.

3. Un disco tiene 131076 bytes de tamaño.

```
static const unsigned MAGIC_SIZE = sizeof (int);

const unsigned SECTORS_PER_TRACK = 32;  ///< Number of sectors per disk
const unsigned NUM_TRACKS = 32;         ///< Number of tracks per disk.
const unsigned NUM_SECTORS = SECTORS_PER_TRACK * NUM_TRACKS;

static const unsigned DISK_SIZE = MAGIC_SIZE + NUM_SECTORS * SECTOR_SIZE;
```

4. MIPS simula 60 instrucciones. Están enumeradas en `machine/encoding.hh`

5. la instrucción `ADD` suma los valores de dos registros, mediante XOR verifica que los registros tengan el mismo signo y, de ser el caso, también el resultado, ya que en caso contrario hubo overflow en la suma. Si no hubo overflow, se guarda el resultado en uno de los registros.

```
case OP_ADD:
  sum = registers[instr->rs] + registers[instr->rt];
  if (!((registers[instr->rs] ^ registers[instr->rt]) & SIGN_BIT)
        && (registers[instr->rs] ^ sum) & SIGN_BIT) {
      RaiseException(OVERFLOW_EXCEPTION, 0);
      return;
  }
  registers[instr->rd] = sum;
  break;
```

6.

Initialize -> `system.cc`

- ASSERT -> `lib/utility.hh`
- SystemDep::RandomInit -> `machine/system_dep.cc`
- atoi -> `stdlib.h`
- debug.SetFlags() -> `lib/debug.cc`
- currentThread->SetStatus() -> `thread/thread.cc`
- interrupt->Enable() -> `machine/interrupt.cc`
- SystemDep::CallOnUserAbort() -> `machine/system_dep.cc`
- SetUp() -> `threads/preemptive.cc`

DEBUG -> `lib/utility.hh`

- Debug::Print()

strcmp -> `string.h`

PrintVersion -> `threads/main.cc`

- printf() -> `stdio.h`

ThreadTest -> `threads/thread_test.cc`

- DEBUG
- strncpy -> `string.h`
- newThread->Fork() -> `threads/thread.cc`
- SimpleThread -> `threads/thread_test.cc`

currentThread->Finish() -> `threads/thread.cc`

- interrupt->SetLevel() -> `machine/interrupt.cc`
- ASSERT
- DEBUG
- GetName() -> `threads/thread.cc`
- Sleep() -> `threads/thread.cc`

7. Simular la CPU simplifica el desarrollo del SO reduciendo tiempos de compilación, ejecución y debugging y permite el uso de debuggers simbólicos.

8. `ASSERT` (definida en `assert.hh`) verifica el estado de un resultado dado y, de ser negativo (`false`), aborta la ejecución del programa y muestra un mensaje de error.

`DEBUG` (definida en `utility.hh`) utiliza la función `Debug::Print()`, que imprime declaraciones de depuraciones.

9.

```
/// * `+` -- turn on all debug messages.
/// * `t` -- thread system.
/// * `s` -- semaphores, locks, and conditions.
/// * `i` -- interrupt emulation.
/// * `m` -- machine emulation (requires *USER_PROGRAM*).
/// * `d` -- disk emulation (requires *FILESYS*).
/// * `f` -- file system (requires *FILESYS*).
/// * `a` -- address spaces (requires *USER_PROGRAM*).
/// * `e` -- exception handling (requires *USER_PROGRAM*).
/// * `n` -- network emulation (requires *NETWORK*).
```

10. Las constantes están definidas en los `Makefile` del proyecto. TODO: Agregar en dónde está definida cada una.

11. `List` es una clase que representa a una lista simplemente enlazada con prioridad, mientras que `SynchList` representa a una lista sincronizada,

esto es, una lista cuyas restricciones son:

- los hilos que quieran eliminar un elemento de la lista esperarán hasta que la lista contenga un elemento.
- Un hilo a la vez puede acceder a la estructura de dato de la lista.

12. `main` se encuentra en los siguientes archivos:

```
threads/main.cc -> main del ejecutable nachos del directorio userprog
bin/disasm.c
bin/fuse/nachosfuse.c
bin/main.c
bin/coff2flat.c
bin/readnoff.c
bin/out.c
bin/coff2noff.c
userland/filetest.c
userland/shell.c
userland/echo.c
userland/halt.c
userland/sort.c
userland/tiny_shell.c
userland/touch.c
userland/matmult.c
```

13. nachOS soporta las siguientes líneas de comandos:

```
nachos [-d <debugflags>] [-p] [-rs <random seed #>] [-z] [-s]
[-x <nachos file>] [-tc <consoleIn> <consoleOut>][-f]
[-cp <unix file> <nachos file>] [-pr <nachos file>] [-rm <nachos file>]
[-ls] [-D] [-tf] [-n <network reliability>] [-id <machine id>]
[-tn <other machine id>]
```

La opción -rs ejecuta `Thread::Yield()` en forma aleatoria y repetitiva.