MINF18
TD OS Reporting
Devoir 0
Members:
Vo Hung Son
Tran Quang Nhat

2. Source code reading

2.1 Simulation principles

Give an example source file of a MIPS user program, and an example source file of the NachOS kernel. What is the programming language used each time?

Answer:

source file of a MIPS user program: Assembly Language

ex:

```
1. .data
2. prompt1: .asciiz "Enter the first number: "
3. prompt2: .asciiz "Enter the second number: "
4. menu: .asciiz "Enter the number associated with the operation you want performed: 1
   => add, 2 => subtract or 3 => multiply: "
5. resultText: .asciiz "Your final result is: "
```

source file of the NachOS kernel: C++ language:

ex:

```
void
Thread::Start (VoidFunctionPtr func, void *arg)
{
        DEBUG ('t', "Starting thread \"%s\" with func = %p, arg = %d\n",
        name, func, arg);

        ASSERT(status == JUST_CREATED);
        StackAllocate (func, arg);

        IntStatus oldLevel = interrupt->SetLevel (IntOff);
        scheduler->ReadyToRun (this);    // ReadyToRun assumes that interrupts
        // are disabled!
        (void) interrupt->SetLevel (oldLevel);
}
```

2.2. System initialization

How is this first kernel thread created?
Answer:
Step 1:Call into Start method.
Step 2: In the Start method call into StackAllocate method
Step 3: In the StackAllocate mehod that we init a stack with size of long
step 4: Register stack by valgrind_id.
Step 5: check constant value to work in range (low addresses to high addresses or opposite)
Step 6: SetupThreadState that it is called be StartupPCState of machine.
Step 7: Register InitialPCState of machine state by func with long value.
Step 8: Register InitialArgState of machine state by arg with long value.
Step 9: Register WhenDonePCState of machine state by ThreadFinish method with long value.
Step 10: Set IntStatus oldLevel = interrupt->SetLevel (IntOff)
Step 11: Run the thread by code line scheduler->ReadyToRun (this);

Where does its stack and its registers come from?
Answer:
It comes from StackAllocate (func, arg) method.

What is the (future) role of the data structure allocated by the instruction:
Answer:
HOST_SNAKE: HP stack works from low addresses to high addresses;HP requires 64-byte frame marker
#else: other archs stack works from high addresses to low addresses
HOST_SPARC:SPARC stack must contains at least 1 activation record to start with.
HOST_PPC
HOST_i386: -4 for the return address
HOST_x86_64: -8 for the return address
HOST_MIPS

Why is it necessary to call the Start method for the next kernel threads? (focus into threads/thread.h and threads/thread.cc)
Answer:
1. Allocate a stack
2. Initialize the stack so that a call to SWITCH will cause it to run the procedure
3. Put the thread on the ready queue

2.3 User program execution
How are the registers of this processor initialized?
Answer:
AddrSpace::InitRegisters
Set the initial values for the user-level register set.

We write these directly into the "machine" registers, so
that we can immediately jump to user code. Note that these
will be saved/restored into the currentThread->userRegisters
when this thread is context switched out.
Step 1: Initial program counter -- must be location of "Start"

Step 2: Need to also tell MIPS where next instruction is, because of branch delay possibility
Step 3: Set the stack register to the end of the address space, where we allocated the stack; but subtract off a bit, to make sure we don't accidentally reference off the end!

What variable is MIPS memory?
Answer:
It is pointer type;
the loading of the program into memory (simulated or real?)
Answer:
Is Real.
What is the name of the exception thrown when an addition (assembly instruction OP_ADD) overflows?

```
 switch (instr->opCode) {

        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(OverflowException, 0);
        return;
   }

  SyscallException,     // A program executed a system call.
                PageFaultException,  // No valid translation found
                ReadOnlyException,   // Write attempted to page marked
                                     // "read-only"
                BusErrorException,   // Translation resulted in an
                                     // invalid physical address
                AddressErrorException, // Unaligned reference or one that
                                     // was beyond the end of the
                                     // address space
                OverflowException,   // Integer overflow in add or sub.
                IllegalInstrException, // Unimplemented or reserved instr.

                NumExceptionTypes
```