# オペレーティングシステム特論

## Advanced Operating Systems
## #1 (2008/10/3)

Fall 2008, Mon. 9:00-10:30, Rm. W831

Instructor: Takuo Watanabe（渡部卓雄）
e-mail: aos08@psg.cs.titech.ac.jp
ext. 3690, office W8E-805
http://www.psg.cs.titech.ac.jp/aos



## Outline

- ➡ Introduction
  - About This Course
  - Course Administrivia
- High-Level Language VMs (1)
  - Implementing Simple Stack VM



## About This Course

- Goal: to give students a better understanding of Virtual Execution Environments（仮想実行環境） and their applications.



## Topics to be Covered (1)

- Process Virtual Machines
  - High-Level Language VMs (HLL-VMs)
    - JVM, Parrot, YARV, etc.
    - WAM, CAM, SECD
  - Other Process-Level VMs
    - Jail, Zap, etc.
- System Virtual Machines
  - VMM, Paravirtualization, Microkernel
    - Xen, Denali, Exokernel, etc.

## Topics to be Covered (2)

- Applications of VMs
  - ex. Secure Computing
    - Terra, Revirt, VM-based Introspection for Intrusion Detection, Collapsar
    - Honeypot, Honeynet Project
    - Monitored Execution



## Prerequisites

- Basic knowledge on:
  - Programming Languages (esp. C)
  - Operating Systems
  - Computer Architectures
  - Logic (First Order Logic)



## Administrativia

- Workload:
  - lectures
  - present at least one paper on VEE
  - report (TBA) on VEE
- Grading:
  - 50% presentation
  - 50% report



## Administrativia

- 60 min. lecture
- 30 min. presentation
  - 20 min. talk + 10 min. discussion
    - Presentation may be either in English or Japanese.
    - Presentation materials (slides and handouts) should be written in English.
  - Reading list is on the course web site. Pick up 3 papers and send me their titles with your preference order until Oct. 9.

## Presentation Guidlines

- Make sure that you understand the paper.
- Prepare some slides.
  - 1 slide ≒ 2 minutes
- Rehearse at least once.
- Split the presentation in three parts.
  - Overview of the paper and explanation of the problem
  - Discussion of key contribution
  - Critique



## Calendar (1)

- 10/3 lecture (1) (introduction)
- 10/10 lecture (2)
- 10/17 lecture (3) + presentation (1)
- 10/31 lecture (4) + presentation (2)
- 11/7 lecture (5) + presentation (3)
- 11/14 lecture (6) + presentation (4)
- 11/21 lecture (7) + presentation (5)
  - (11/28 no class)
- 12/5 lecture (8) + presentation (6)



## Calendar (2)

- 12/12 lecture (9) + presentation (7)
- 12/19 lecture (10) + presentation (8)
- 1/9 lecture (11) + presentation (9)
- 1/23 lecture (12) + presentation (10)
  - or presentations (10-12)
- 1/30 presentations (11-13)
  - or presentations (13-15)
- (2/6 presentations)



## Lecture Plan (1)

- Introduction
- High Level Language VMs (1)
  - Interpreters
    - Simple Bytecode Interpreter
  - Various Improvements of Interpreters
    - Threaded Code, Stack Caching, Superoperators

## Lecture Plan (2)

- High-Level Language VMs (2)
  - Java VM
    - Bytecode Verification, JIT
  - VM for other languages: CAM, WAM, etc.
- Runtime Systems
  - Memory Management
- Process VMs
- System VMs
  - VMM, Paravirturalization



## Books

- Virtual Machines
  - Iain D. Craig
  - Springer, 2006 (1-85233-969-1).
- Virtual Machines: Versatile Platforms for Systems and Processes
  - James E. Smith & Ravi Nair
  - Morgan Kaufmann, 2005 (1-55860-910-5).



## Outline

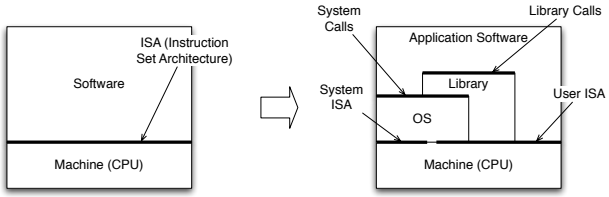
- Introduction
  - About This Course
  - Course Administrivia
- ➡ High-Level Language VMs (1)
  - Implementing Simple Stack VM in C



## Virtual Machine

- Classic Definition
  - "an efficient, isolated duplication of a real machine"
    - G. J. Popek & R. P. Goldberg, "Formal Requirements for Virtualizable Third Generation Architectures", Comm. ACM, 17(7), 1974.
- Current
  - a software that provides virtual software execution environment
    - not restricted to real machines
  - Taxonomy
    - System VM
    - Process VM

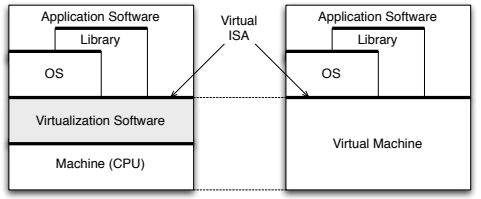
# Machine Interfaces




Application Binary Interface (ABI)
= System Calls + User ISA
Application Program Interface (API)
= System Calls + Library Calls + User ISA

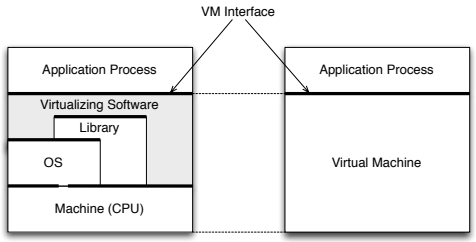


# System Virtual Machine






# Process Virtual Machine






# High Level Language VM

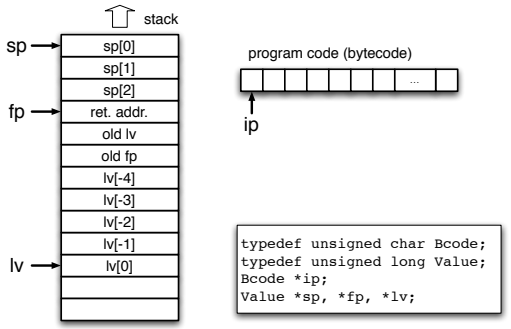
- An interpreter for virtual (abstract) machine instructions
  - a kind of process VMs
- Constructions
  - Execution Mechanisms
    - Stack Machines
      - JVM, .Net CLR, etc.
    - Register Machines
      - CAM, Parrot, etc.
  - Instruction Representation
    - Bytecode
    - Wordcode, VLIW-like code, etc.

# Example: A Simple VM in C

- Bytecode Machine
  - Java-like instruction set
- Stack Machine
  - Function arguments and local variables are on the value stack.
  - 32-bits value
    - integer
    - function pointer
- Source code can be obtained from the course website.



# VM Structure






# Instructions

- `nop, dup`
- `ifeq, ifne, iflt, ifle, ifgt, ifge`
- `giload, gaload, gistore, gastore`
- `iload, aload, istore, astore`
- `bipush, sipush, iconst_m1, iconst_0, iconst_1, iconst_2`
- `iadd, isub, imul, idiv, irem`
- `icall, acall, ticall, tacall, sicall, sacall`
- `iret, aret`
- `goto, halt`



# The Main VM Function

```
void vm (Bcode code[], size_t codelen,
         Value *vs_base, size_t vs_size,
         value gv[], size_t gv_size,
         vm_mode mode) {
    Bcode *ip = code;       // instruction pointer
    Value *sp = vs_base;    // stack pointer
    Value *fp = sp;         // frame pointer
    Value *lv = sp;         // local vars base pointer

    for (;;) {
        switch (*ip) {
          // giant switch implementing instructions
        }
    }
}
```

# Fetch-Decode-Execute (Giant Switch)

```
for (;;) {
    switch (*ip) {
    case i_nop:
        ip++;
        break;
          :
    case i_iadd:
        sp[1] = sp[1] + sp[0];
        sp++;
        ip++;
        break;
          :
    }
}
```



# Implementing Instructions

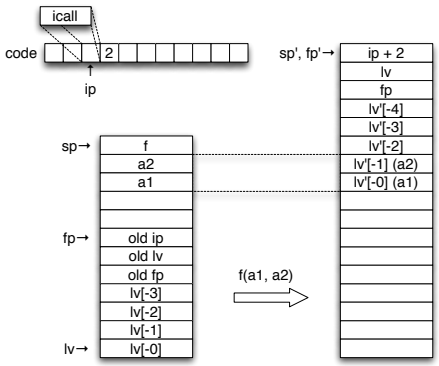
```
iadd:
    sp[1] = sp[1] + sp[0];
    sp++;
    ip++;
```

```
iload:
    sp--;
    sp[0] = lv[-(int)ip[1]]
    ip += 3;
```

```
goto:
    ip += ((signed char)ip[1] << 8 | ip[2]) + 3;
```



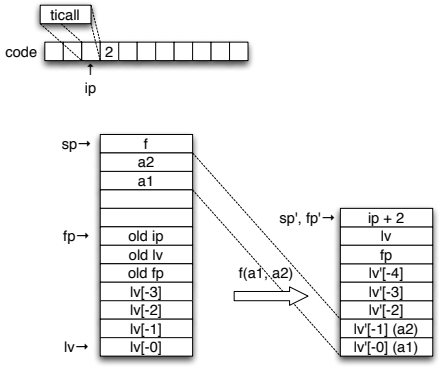
# Function Call




# Function Call Implementation

```
icall:
    Function f = (Function)sp[0];
    int nargs = (int)ip[1];
    Value *newfp = sp - f->nlv - 2;
    newfp[0] = (Value)(ip + 2);
    newfp[1] = (Value)lv;
    newfp[2] = (Value)fp;
    lv = sp + nargs;
    sp = fp = newfp;
    ip = f->code;
```

# Tail-Call



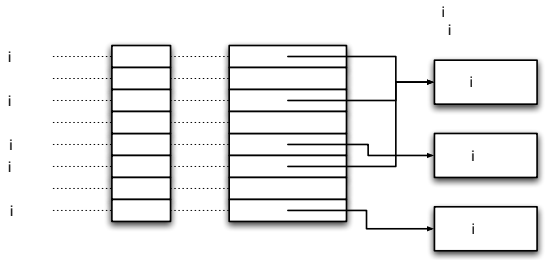



# Tail-Call Implementation

```
ticall:
    Function f = (Function)sp[0];
    int nargs = (int)ip[1];
    Value ip_save = fp[0];
    Value lv_save = fp[1];
    Value fp_save = fp[2];
    Value *arg = sp + nargs;
    int i;
    for (i = 0; i < nargs ; i++)
        lv[-i] = arg[-i];
    fp = lv - nargs - f->nlv - 3;
    sp = fp;
    fp[0] = ip_save;
    fp[1] = lv_save;
    fp[2] = fp_save;
    ip = f->code;
```



# Threaded Code






# Direct Threaded Code (DTC)

```
iadd:
  sp[1] = sp[1] + sp[0];
  sp++;
  ip++;
  goto tcode[ip];
```

## Labels as Value Extension in GCC

```
void *laddr;

laddr = &&label;

goto *laddr;

label:
    ....
```



## DTC in GCC

```
#define NEXT goto **ip

typedef void *Tcode;

Tcode tcode[] = ...; // Program in DTC
Tcode *ip = tcode;
NEXT;

l_nop:
  ip++;
  NEXT;
    :
l_iadd:
  sp[1] = sp[1] + sp[0];
  sp++;
  ip++;
  NEXT;
    :
```



## vmbench.c

- Result
  - svm (Giant Switch)
    - 14.96 sec (PPC), 4.27 sec (IA32)
  - tvm (DTC)
    - 10.85 sec (PPC), 4.78 sec (IA32)
- Environment (OS X 10.4.10, GCC 4.0.1):
  - PPC: 1.25GHz PPC G4 , 2GB RAM
  - IA32: 2.33GHz Core2 Duo, 2GB RAM



## Summary

- Course Guidance
- Implementing Simple Stack VM in C