COL380

Introduction to
Parallel & Distributed Programming

2-0-2

Subodh Kumar

Agenda

- OS basic
- A review of Architecture

Some OS Basics

- Process & thread
  - ➡ Scheduling, Context-switching and concurrency
  - ➡ User space, Kernel space
  - ➡ System calls
- Address space & Name space
- Virtual Memory
  - ➡ Caches
- Synchronization

- ✦ Shell (bash)
- ✦ time
- ✦ PMU (perf)

Execution Steps

edit file.c:

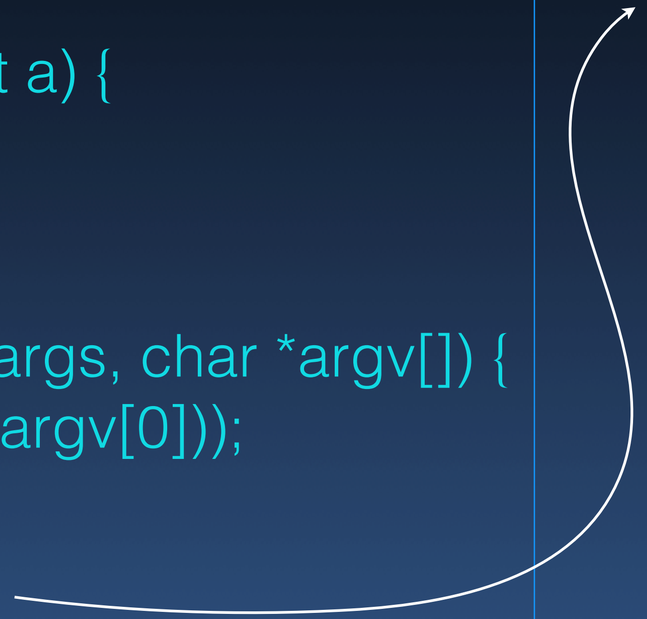
```
#include …
void func(int a) {
    sleep(a);
}

int main(int args, char *argv[]) {
    func(atoi(argv[0]));
    …
    return v;
}
```

gcc file.c -o exec

time ./exec

echo $?

```
movl  %eax, -4(%rbp)
```



Execution Steps

edit file.c:

```
#include …
void func(int a) {
    sleep(a);
}

int main(int args, char *argv[]) {
    func(atoi(argv[0]));
    …
    return v;
}
```

gcc file.c -o exec

time ./exec

echo $?

```
movl  %eax, -4(%rbp)
```

1. Map Virtual addr ➜ Physical
2. Fetch: L1, L2 .., Memory

Execution Steps

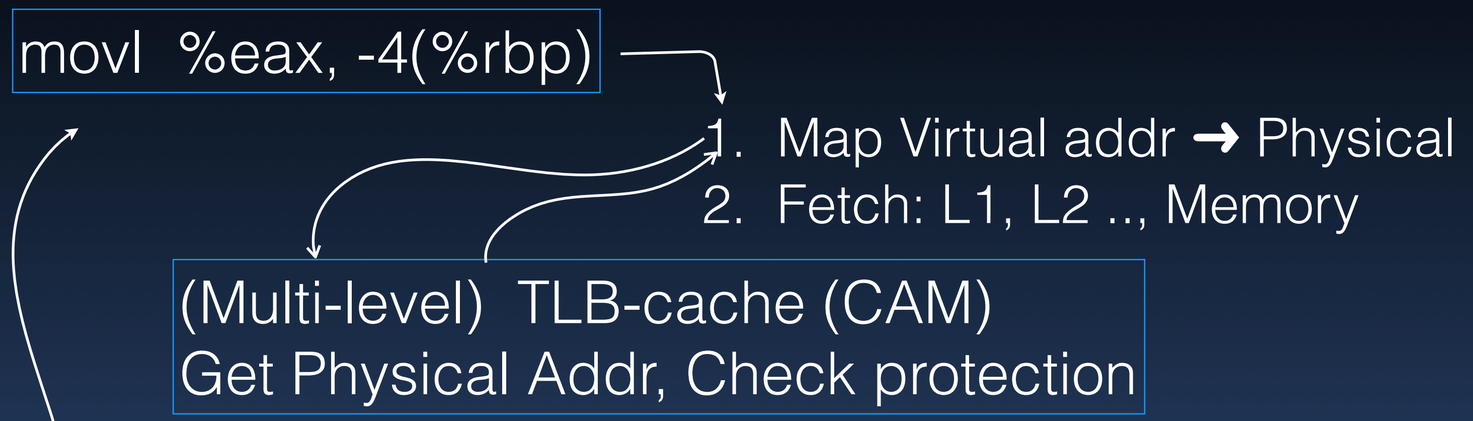
edit file.c:

```
#include …
void func(int a) {
    sleep(a);
}

int main(int args, char *argv[]) {
    func(atoi(argv[0]));
    …
    return v;
}
```

gcc file.c -o exec

time ./exec

echo $?



Execution Steps

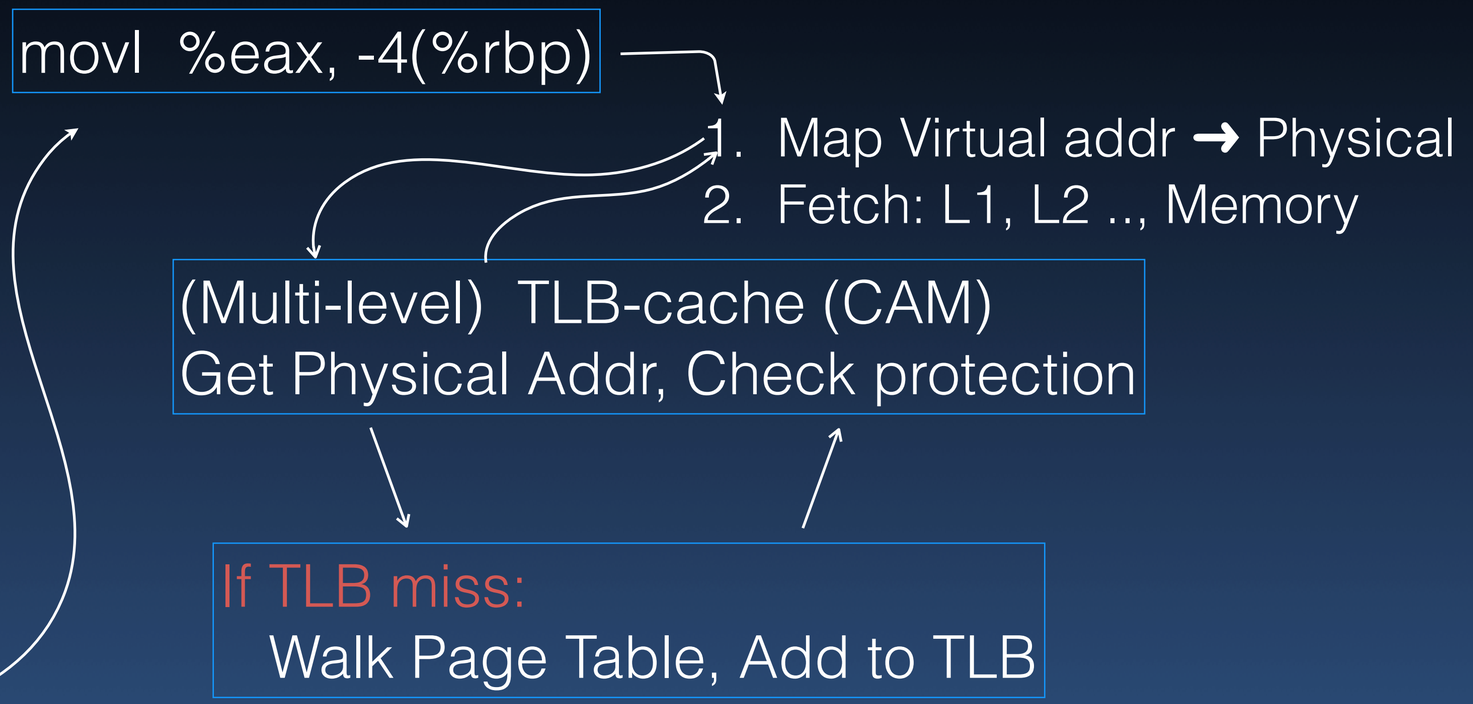
edit file.c:

```
#include …
void func(int a) {
    sleep(a);
}

int main(int args, char *argv[]) {
    func(atoi(argv[0]));
    …
    return v;
}
```

gcc file.c -o exec

time ./exec

echo $?



Execution Steps

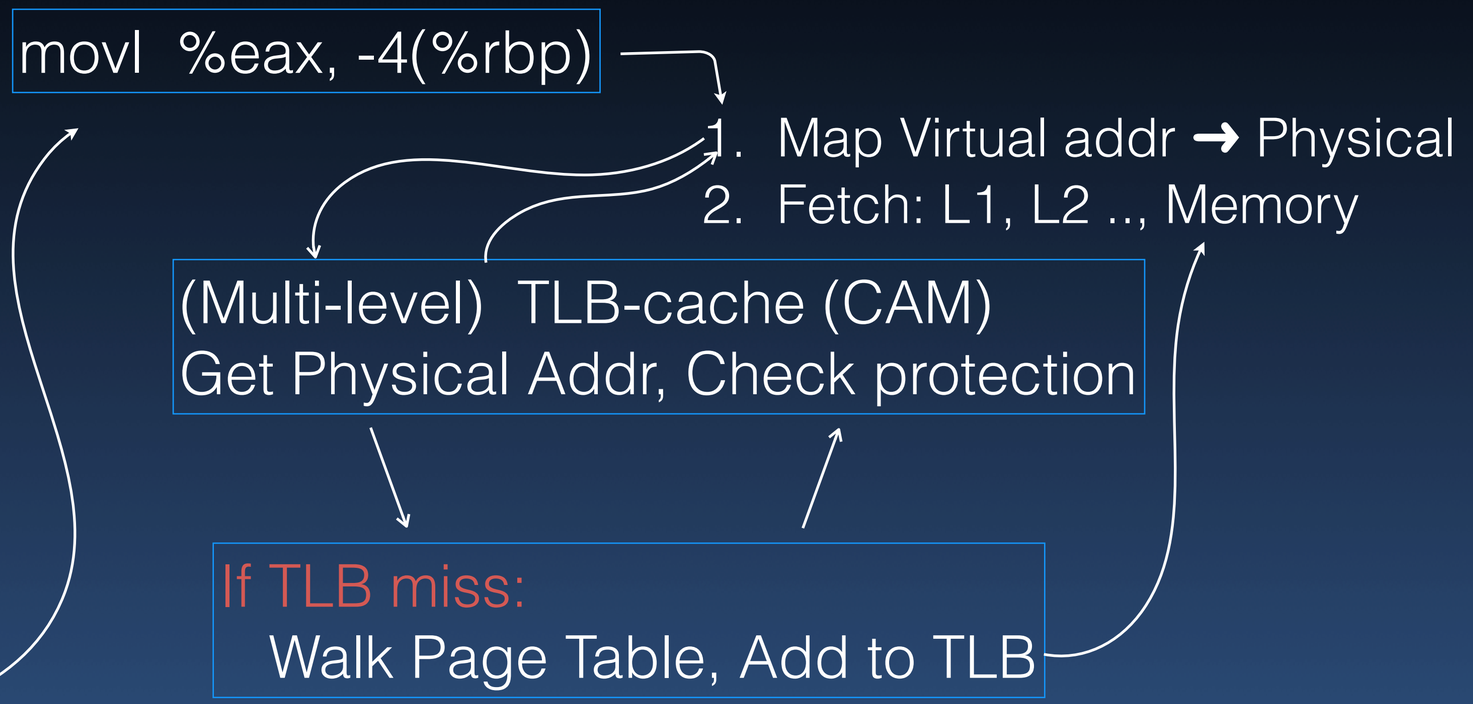
edit file.c:

```
#include …
void func(int a) {
    sleep(a);
}

int main(int args, char *argv[]) {
    func(atoi(argv[0]));
    …
    return v;
}
```

gcc file.c -o exec

time ./exec

echo $?



Execution Steps

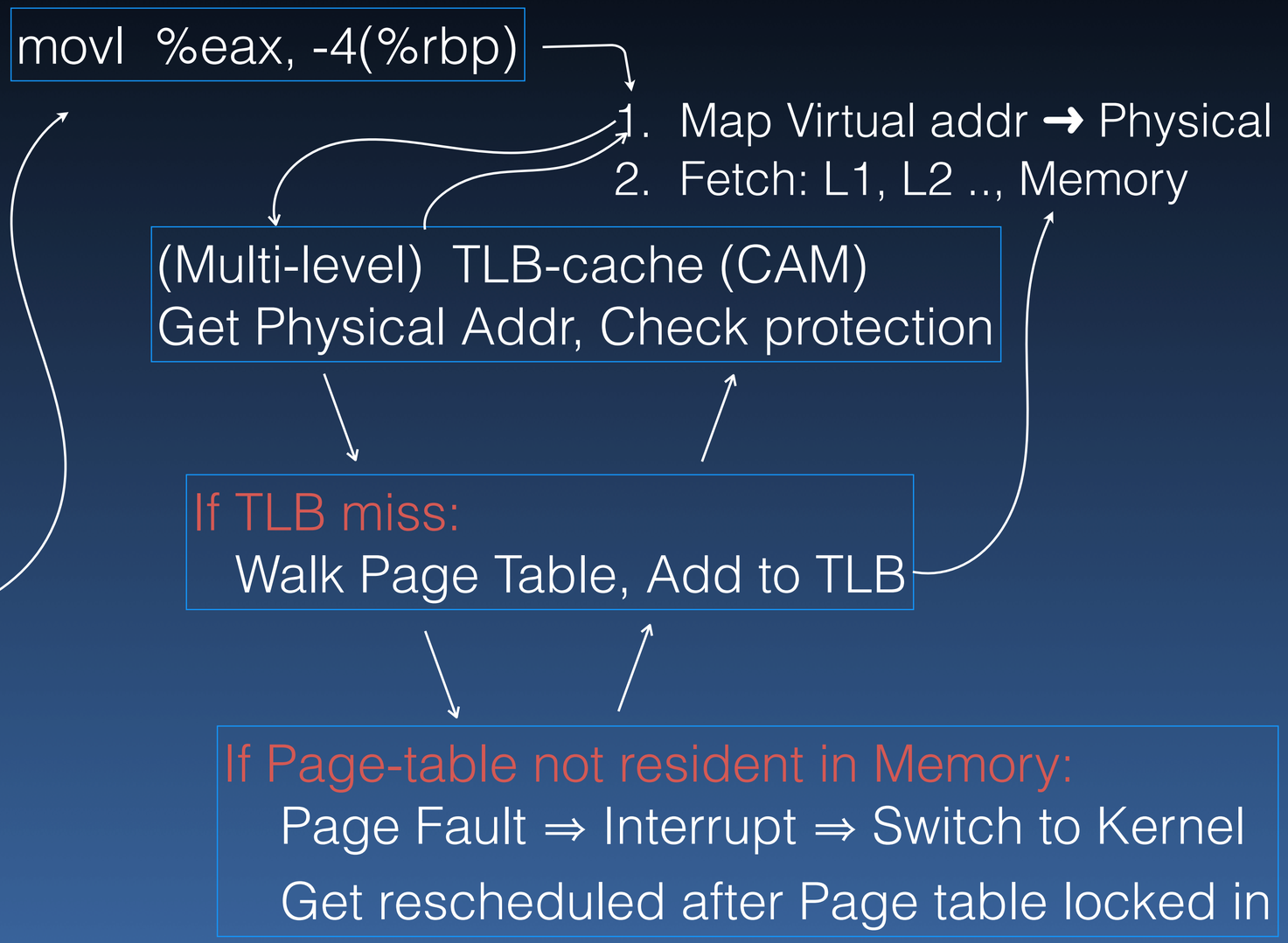
edit file.c:

```
#include …
void func(int a) {
    sleep(a);
}

int main(int args, char *argv[]) {
    func(atoi(argv[0]));
    …
    return v;
}
```

gcc file.c -o exec

time ./exec

echo $?



Execution Steps

edit file.c:

```
#include …
void func(int a) {
    sleep(a);
}

int main(int args, char *argv[]) {
    func(atoi(argv[0]));
    …
    return v;
}
```

gcc file.c -o exec

time ./exec

echo $?

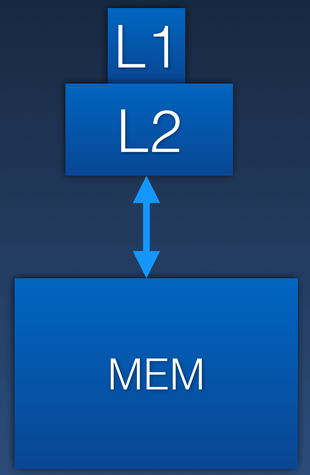
movl %eax, -4(%rbp)

1. Map Virtual addr ➜ Physical
2. Fetch: L1, L2 .., Memory

(Multi-level) TLB-cache (CAM)
Get Physical Addr, Check protection

If TLB miss:
Walk Page Table, Add to TLB

If Page-table not resident in Memory:
Page Fault ⇒ Interrupt ⇒ Switch to Kernel
Get rescheduled after Page table locked in



Execution Steps

edit file.c:

```
#include …
void func(int a) {
    sleep(a);
}

int main(int args, char *argv[]) {
    func(atoi(argv[0]));
    …
    return v;
}
```

gcc file.c -o exec

time ./exec

echo $?

movl %eax, -4(%rbp)

1. Map Virtual addr ➜ Physical
2. Fetch: L1, L2 .., Memory

(Multi-level) TLB-cache (CAM)
Get Physical Addr, Check protection

If TLB miss:
Walk Page Table, Add to TLB

If Page-table not resident in Memory:
Page Fault ⇒ Interrupt ⇒ Switch to Kernel
Get rescheduled after Page table locked in

Execution Steps

edit file.c:

```
#include …
void func(int a) {
    sleep(a);
}

int main(int args, char *argv[]) {
    func(atoi(argv[0]));
    …
    return v;
}
```

gcc file.c -o exec

time ./exec

echo $?

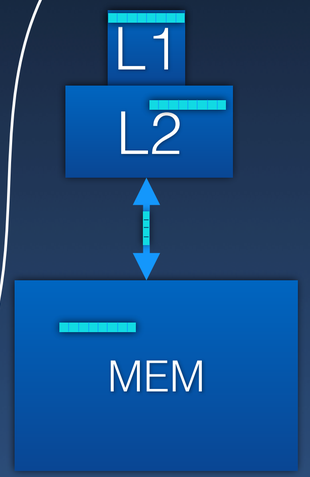
movl %eax, -4(%rbp)

1. Map Virtual addr ➜ Physical
2. Fetch: L1, L2 .., Memory

(Multi-level) TLB-cache (CAM)
Get Physical Addr, Check protection

If TLB miss:
Walk Page Table, Add to TLB

If page not resident in Memory:
Page Fault ⇒ Interrupt ⇒ Switch to Kernel
Get rescheduled after Page table locked in



It's the Memory Stupid!

- Accessing variables is much 'slower' than computing
- Memory sizes are often sufficient
  - ➡ Particularly, when distributed among many nodes
    - ▸ Out-of-core computation needed sometimes (we won't focus)
- Caches are small
  - ➡ Not very helpful if traversing large swaths of memory
  - ➡ But, mind the line
    - ▸ And, prefetching is prevalent
  - ➡ Beware of sharing across caches, particularly false-sharing

Cache Example

```
for(int i=0; i<n; i++) {
     a[i] = b[i] + c[i];
}
for(int i=0; i<n; i++) {
    d[i] = e[i] + f[i];
}
```

vs

```
for(int i=0; i<n; i++) {
     a[i] = b[i] + c[i];
     d[i] = e[i] + f[i];
}
```

```
for(int i=0; i<N; i++) {
    d[i] = x;
}
```

```
for(int i=0; i<N*STEP; i++) {
    d[(i*STEP)] = x;
}
```

Subodh Kumar

Review

- Processes
- Address spaces
- Virtual memory
- Caches