CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Contiguous Memory Allocation

Reminder: we want to allocate memory for various processes (and the OS) in a way that is efficient

Simplest idea was swapping -- easy because just one process in memory at once, but context switch time was unacceptably slow

We will need to have more than one process actually in memory at once

OS resides in a fixed part of memory (usually the low addresses)

Rest of memory must be split up somehow among various processes

Need to be as efficent as possible to make effective use of available memory

Most straightforward idea is probably **contiguous memory allocation**

Contiguous memory allocation -- each process resides in its own single section of memory, and successive sections are contiguous

How can memory be protected in such a system?

We have already seen the mechanisms we need: a relocation register and a limit register

We will combine them in a slightly different way than previously seen

Relocation register contains the smallest physical address (the start of physical memory for this process)

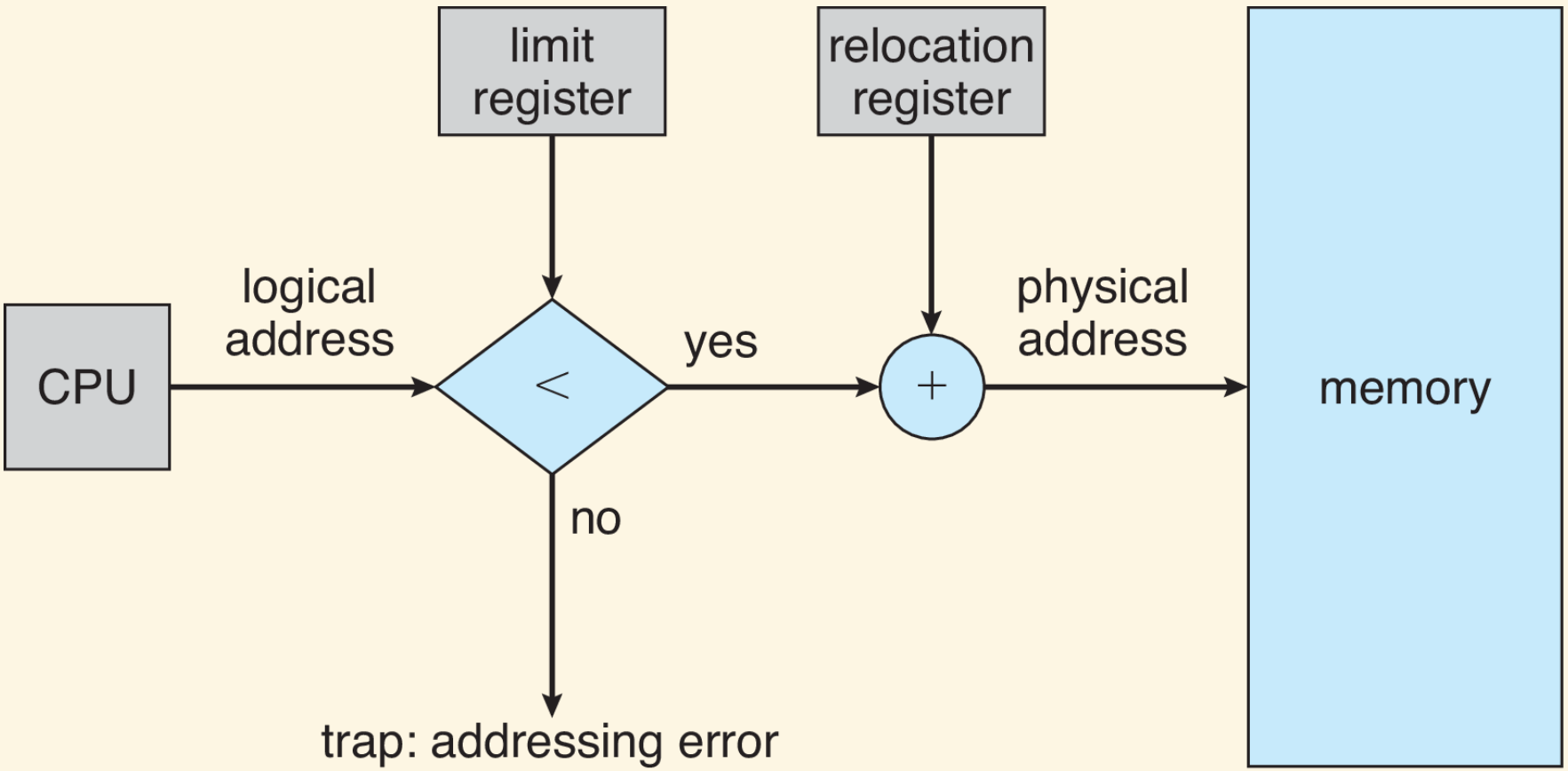
Limit register contains largest logical address (size of physical memory for this process)

For relocation register value R and limit register value L,

Logical address range is `[0, L]`

Physical address range is `[R, R + L]`

As always, MMU handles address mapping


limit
register
relocation
register
logical
address
CPU
<
yes
+
physical
address
memory
no
trap: addressing error

As before, OS must control relocation and limit registers

In a sense, protection mechanism is a two-for-one deal

Protects OS and other processes, and also handles mapping of logical addresses to physical ones

Contiguous memory allocation

Simplest way to allocate memory in this scheme is **fixed partitions**

Maximum number of processes in memory determined by number of partitions

Downside?

Small processes take up more space than they need

Large processes may not be able to run

This scheme is no longer in use

In a **variable-partition** scheme, the size of each partition can be different from the sizes of the others

OS keeps table tracking which memory is available

Region of available memory is called a **hole**

Initially, all of memory (besides OS section) is one large hole

When process enters the system, OS will assign it to a hole based on its memory requirements

When process exits, memory is reclaimed by OS

Table in OS is essentially monitoring a set of holes

Process will not necessarily be assigned entire hole, but rather only what it needs

When process exits and hole is freed, it can be merged with adjacent holes to form larger hole

OS keeps a queue of processes waiting to enter memory (**input queue**)

If no hole is large enough for first process in queue, OS can either wait or move on to another process in the queue

Input queue has its own scheduling algorithm, like ready queue for CPU, which we will not investigate

Investigate next time how OS chooses which hole to assign process