CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Frame Allocation

Despite having large amount of virtual memory, system still has fixed amount of physical memory

How best to allocate limited memory among processes?

Consider first case of one process

Assume system has 128 KB split into 1 KB pages

OS takes (for example) 35 pages

Leaves 93 KB (i.e., 93 frames) for process

OS will simply give user free frame every time page fault occurs until memory is full (all frames allocated)

At that point, OS pages out old page whenever new page must be read in

Can use any strategy already discussed (LRU, FIFO)

Could also perform optimizations such as keeping free frame pool

When considering more than one process, first thing we do will be to determine bounds

Upper bound for system is obvious -- cannot exceed number of frames

Lower bound for processes also exists

When instruction encounters page fault, it must be restarted after OS services page fault

To do so, instruction must still be in memory in addition to newly read-in page(s)

If more than one page can be accessed in single instruction, must ensure process has at least as many frames as possible pages accessed in single instruction

Some architectures allow 6 pages, 8 pages, or even entire memory space to be touched by single instruction

Between minimum (mandated by architecture) and maximum (mandated by physical memory), a lot of leeway exists

Simple ways to decide are **equal allocation** and **proportional allocation**

Equal allocation is when all processes receive same number of frames

Remainder can go into free frame pool

Simple, but can be wasteful

If one process needs just 10 frames but is assigned 40, remaining 30 frames could be put to better use elsewhere

Proportional allocation is allocating frames according to process size

Assume:

- 93 free frames
- Process A: 100 KB
- Process B: 200 KB

Process A allocated (100/300)*93

Process B allocated (200/300)*93

In addition to size, priority may be considered

Regardless of how allocation is portioned out, more multiprogramming means smaller allocations

OS attempts to keep multiprogramming as high as possible to improve resource utilization

Another consideration is global vs local replacement

In **local replacement**, frame allocation for a process is constant, regardless of any increase in need. Frames to satisfy page fault will be pulled from those already belonging to faulting process

In **global replacement**, process may be assigned frame currently in use by other process, allowing receiving process to grow as demand rises

Global replacement generally does better job allocating memory frames to processes that will make use of them

Typically results in higher throughput for system

Global replacement the more common choice for actual systems