CIS 452 - Operating Systems Concepts

Nathan Bowman

Images taken from Silberschatz book

Other Paging Considerations

Main issues in paging system are

- page replacement algorithm
- frame allocation policy

A few other miscellaneous issues to be considered

Prepaging

In pure demand paging, pages brought into memory only when required (page fault)

Prepaging -- predicting which pages will be needed and paging them in ahead of time

Prepaging cannot save data transfer, but can save on page faults

Prepaging results in wasted time for reading in pages if those pages are not needed

Whether prepaging is worthwhile will depend on

- page fault time
- accuracy of prepaging

One use case is dealing with pages that have been swapped out and must be swapped back in

OS can keep track of working set of pages and record it when process is swapped out

When process is swapped back in, OS can page in entire working set in case pages are likely to be used again

Page Size

Many tradeoffs involved in choosing page size

Page size is choice for machine designers, not OS designers

Page sizes are always powers of 2, generally between 2^12 (4,096) and 2^22 (4,194,304) bytes

Smaller page sizes

Less internal fragmentation

Each page fault is less expensive

Less unused data will be read in, resulting in less overall data transfer

Larger page sizes

Fewer total pages and therefore smaller page tables, which can be large savings in memory

Fewer page faults will occur

Page faults more expensive, but one large page fault can be less expensive than several small ones

In general, page fault sizes have tended to grow over time

TLB Reach

TLB must have high hit ratio to be effective

Different, but related metric is TLB reach

TLB reach is amount of memory accessible from TLB

Calculated as (# entries)*(page size)

If size of working set is larger than TLB reach, large number of TLB misses will occur

Larger page sizes can help with this

Some systems support multiple page sizes, in which case TLB must be managed by OS

Page locking

Sometimes a page should not be removed from memory for reasons of efficiency or correcness

Such a page can be **locked** by OS to ensure it will not be paged out

For example, some or all of OS memory is typically locked

Page belonging to user process may be locked in case of I/O, otherwise following error could occur

- User process requests I/O and gives location of buffer to read into
- Context switch occurs, followed by page fault
- Page containing buffer paged out, new page read into same frame
- I/O completes, reads data into location now owned by different process

OS may also allow user programs to lock page into memory for efficiency reasons

In this case, lock is called **pinning**

Pinning must be used cautiously -- if OS allows too many pages to be pinned, poor system performance may result

OS may allow user code to *request* pinning without guaranteeing that request will be granted