Working set model
Process’ behavior

• Locality of reference: most of the time the last $k$ references are within a finite set of pages < a large address space
• The set of pages a process is currently using is called the *working set* of the process
• Knowing the working set of processes we can do very sophisticated things (e.g. pre-paging)
Definition of working set

We define the working set of information $W(t, \tau)$ of a process at time $t$ to be the collection of information referenced by the process during the process time interval $(t-\tau, t)$.

Fig. 2. Definition of $W(t, \tau)$
More definitions

• \( \tau \) is the working set parameter

• The elements of \( W(t, \tau) \) are pages but they can be anything else used by a process (any unit of information)
Working set size

- \( w(t, \tau) = \text{number of pages in } W(t, \tau). \)

![Graph showing the behavior of \( \omega(t, \tau) \)]

**Fig. 3.** Behavior of \( \omega(t, \tau) \)
Properties of the WS

• P1, size,
  – \( w(t,0)=0 \)
  – Monotonically increasing & concave downward (see previous slide)

• P2, prediction
  – \( W(t, \tau) \) is a good predictor of \( W(t+\alpha, \tau) \) for \( \alpha<\tau \)
Benchmarking the WS

• P3, reentry rate

\[ \lambda(\tau) = \frac{1 - F_x(\tau)}{\bar{x}} \]

where \( F_x(\tau) = \Pr\{x \leq \tau\} \) and 
\[
\bar{x} = \int_0^\infty t \cdot f_x(t) \, dt
\]

Average process time between references to same page

\( T \) is the time required to transfer a page from auxiliary memory (disk) to main memory, \( \Phi \) is the traffic rate (total)

\[ \Phi(\tau) = \frac{\beta M \lambda(\tau)}{1 + \lambda(\tau)T} \]

where \( \beta \) fraction of pages used by all WS

\( M \) is the total number of pages

This is an equilibrium situation which assumes some traffic which is bidirectional, thus the bus should have a bandwidth of at least \( 2\Phi(\tau) \).
Benchmarking the WS

• P4, sensitivity to changes in $\tau$

$$\sigma(\tau) = -\frac{d}{d\tau} \lambda(\tau) = \frac{f_x(\tau)}{\bar{x}}$$

That is, if $\tau$ is decreased by $d\tau$, $\lambda(\tau)$ increases by $\sigma(\tau) \, d\tau$

$\sigma(\tau) \geq 0$; reducing $\tau$ can never result in a decrease in the reentry rate $\lambda(\tau)$. 
Choice of $\tau$

• To choose $\tau$ we need to consider:
  – $M$
  – $T$
  – When $\tau$ is small, check $\Phi$ (estimate starting from $x$) to make sure we’re in a reasonable range
  – When $\tau$ is large, need to determine this starting from the desired number of WS in memory
Residency

• How long a page is in memory
Measuring the WS

• In theory it can be done in HW with timers
• In practice (software), sampling the page table

\[ \text{sample} = \frac{\tau}{K} \]

K determines the grain of the sampling
Implementation

As indicated by Figure 5, each page table entry contains an "in-core" bit $M$, where $M = 1$ if and only if the page is present in main memory. It also contains a string of use bits $u_0, u_1, \ldots, u_k$. Each time a page reference occurs $l \rightarrow u_0$. At the end of each sampling interval $\sigma$, the bit

![Diagram of page table entry]

TYPICAL PAGE TABLE ENTRY

SHIFT AT END OF SAMPLING INTERVAL

Fig. 5. Page table entries for detecting $W(t, K\sigma)$
Implementation

pattern contained in $u_0, u_1, \ldots, u_K$ is shifted one position, a 0 enters $u_0$, and $u_K$ is discarded:

$$u_{K-1} \rightarrow u_K$$

$$\vdots$$

$$u_0 \rightarrow u_1$$

$$0 \rightarrow u_0.$$  \hspace{1cm} (15)

Then the logical sum $U$ of the use bits is computed:

$$U = u_0 + u_1 + \cdots + u_K,$$ \hspace{1cm} (16)

so that $U = 1$ if and only if the page has been referenced during the last $K$ sampling intervals; of all the pages associated with a process, those with $U = 1$ constitute its working set $W(t, K \sigma)$. If $U = 0$ when $M = 1$, the page is no longer in a working set and may be removed from main memory.
WS based algorithm

• Store time information in the table entries
• At clock interrupt handle R bits as usual (clear them)
• At page fault, scan entries:
  – If R=1 just store current time in the entry
  – If R=0 compute “current-last time page was referenced” and if > threshold the page can be removed since it’s no longer in the working set (not used for threshold time)
• **Note**: we’re using time rather than actual memory references
**WSClock algorithm**

- Use the circular structure (as seen earlier)
- $R=1$, page in the WS – don’t remove it
- $R=0$, $M=0$ no problem (as before)
- $M=1$, schedule disk write appropriately to procrastinate as long as possible a process switch
  - No write is schedulable ($R=1$ always), just choose a clean page
Segmentation
Why?

- Many separate address spaces (segments) (e.g. data, stack, code, and many others if needed)
- Each segment is separate (e.g. addresses from 0 to some MAX)
- Segments might have different lengths
- Segment number + address within segment
- Linking is simplified (libraries within different segments can assume addresses starting from 0) – e.g. if a part of the libraries is recompiled the remainder of the code is unaffected
- Shared library (DLL’s) implementation is simpler (the sharing is simpler)
### Comparing paging and segmentation

<table>
<thead>
<tr>
<th>Consideration</th>
<th>Paging</th>
<th>Segmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Need the programmer be aware that this technique is being used?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>How many linear address spaces are there?</td>
<td>1</td>
<td>Many</td>
</tr>
<tr>
<td>Can the total address space exceed the size of physical memory</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Can procedures and data be distinguished and separately protected?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Can tables whose size fluctuate be accommodated easily?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Is sharing of procedures between users facilitated?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Why was this technique invented?</td>
<td>To get a large linear address space without having to buy more physical memory</td>
<td>To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection</td>
</tr>
</tbody>
</table>
Pure segmentations
Segmentation + paging (Pentium)

- 16K segments
- 1G 32bit words (DoubleWords)
- Two tables: LDT, GDT – Local (to the process) and global (to the processor) descriptor table
- To work with a segment the machine loads the segment number into a special register (CS, DS, etc.) – CS, DS are 16 bit registers
- The descriptor of the segment (see next slide)
The segment descriptor

- This is used by the microcode within the Pentium to work with segments

- Limit in pages/bytes
- 16/32 bit segment
- Privilege level
- Segment type protection
- Segment present in memory
- System/application
- Limit (20 bits)
- Page size is 4K

- Base 24-31
- G
- D
- 0
- Limit 16-19
- P
- DPL
- S
- Type
- Base 16-23
- Base 0-15
- Limit 0-15

- CS/DS
- Index
- G/L
- Privilege

OS 2007-08
Getting the address

Selector

Descriptor

Base address
Limit
Other fields

Offset

32-bit linear address

+
Paging on the Pentium

- 2-level page table in memory

<table>
<thead>
<tr>
<th>Dir</th>
<th>Page</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 bits</td>
<td>10 bits</td>
<td>12 bits</td>
</tr>
</tbody>
</table>

Each points to 4Mbytes of pages
More on the Pentiums

- TLB, to avoid repeated accesses to memory
- The whole thing can be used with just a single segment to obtain a linear 32bit address space
- Set base and limit appropriately
- Protection (a few bits)