Avoiding OOM (Out of Memory) on Embedded Linux

April 15, 2008

YoungJun Jang yj03.jang@samsung.com

Software Laboratories

Samsung Electronics

Contents

Problem statement

- What's the problem of this code?

- On Linux system
- 64M RAM available

```
main(void)
{
    char *p;
    for (i=0; i<100; i++)
    {
        p = malloc( 1M );
        if ( p == NULL )
        {
            // error handling
        }
    }
}
```

[CODE 1]

Problem statement (cont.)

- Then how about that code?

- On Linux system
- 64M RAM available

```
main(void)
{
   char *p;
   for (i=0; i<100; i++)
   {
      p = malloc( 1M );
      if ( p == NULL )
      {
         // error handling
      }

      memset( p, 0, 1M );
   }
}
```

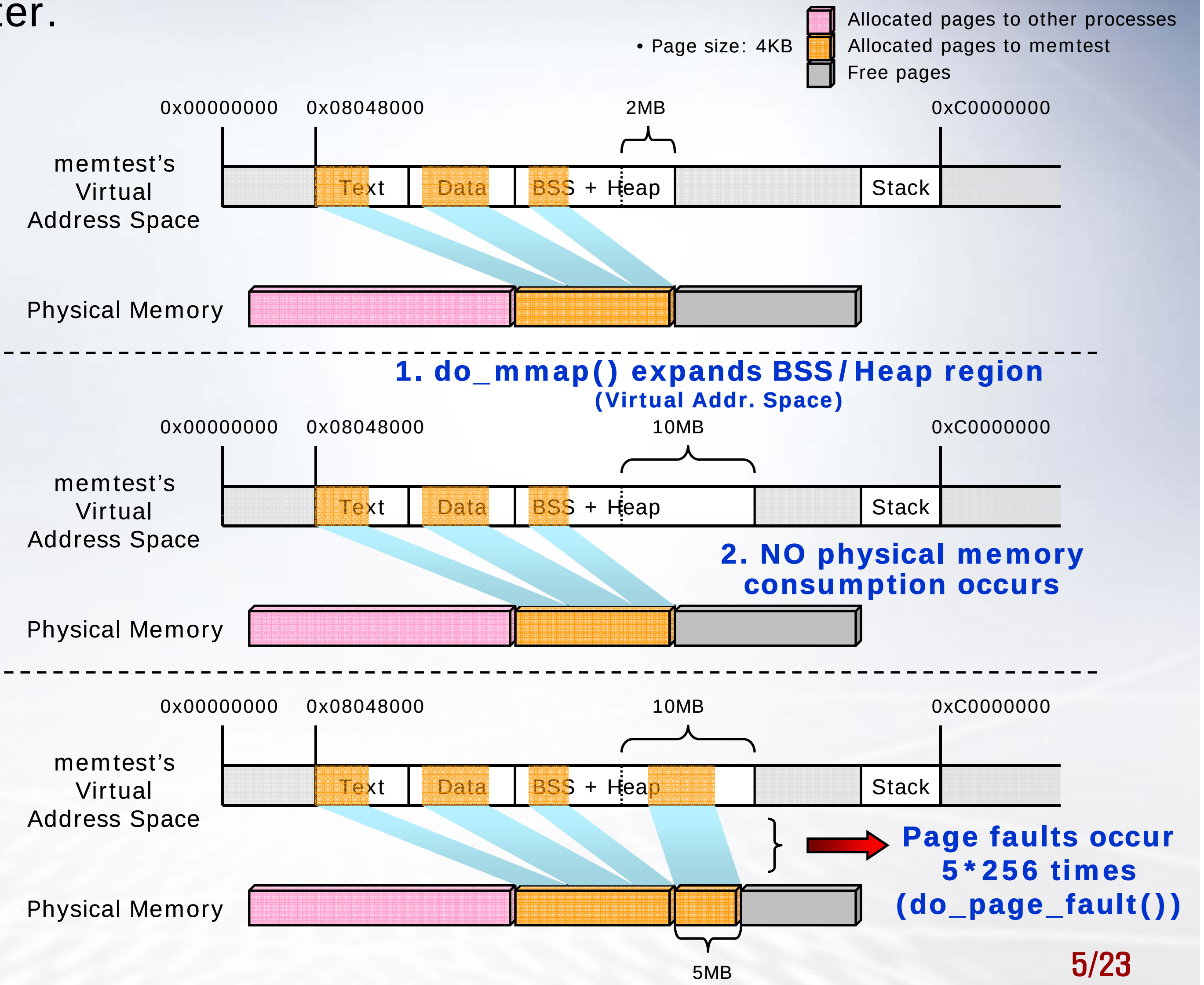
[CODE 2]

Demand Paging

- Separated virtual and physical address
- Reserve virtual address first, and allocate physical memory when accessing virtual memory later.

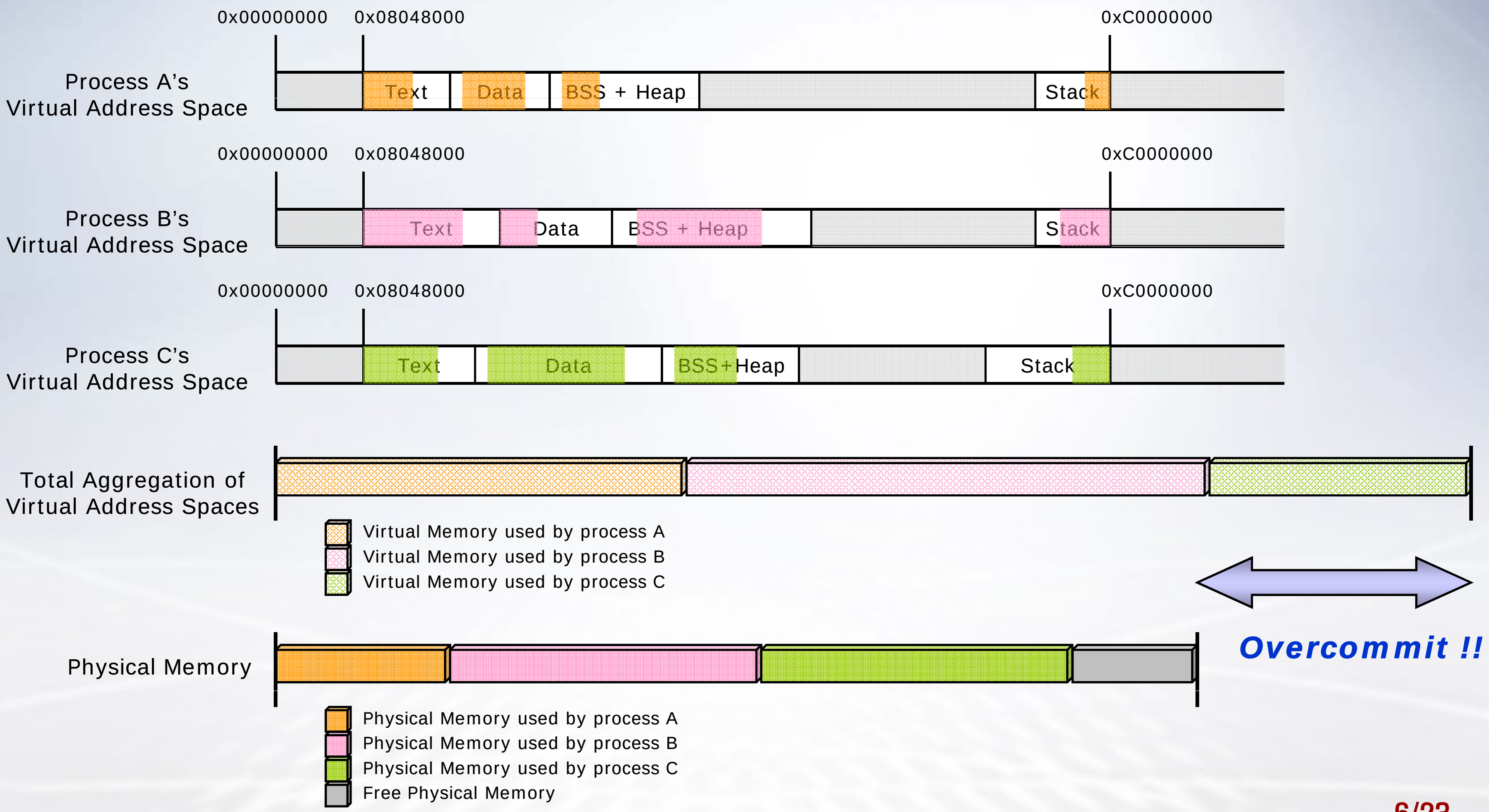
```
main(void)
{
. . .

char *p;
p = malloc(10M);

. . .

memset( p+2M, 0, 5M );

. . .
```

memtest.c



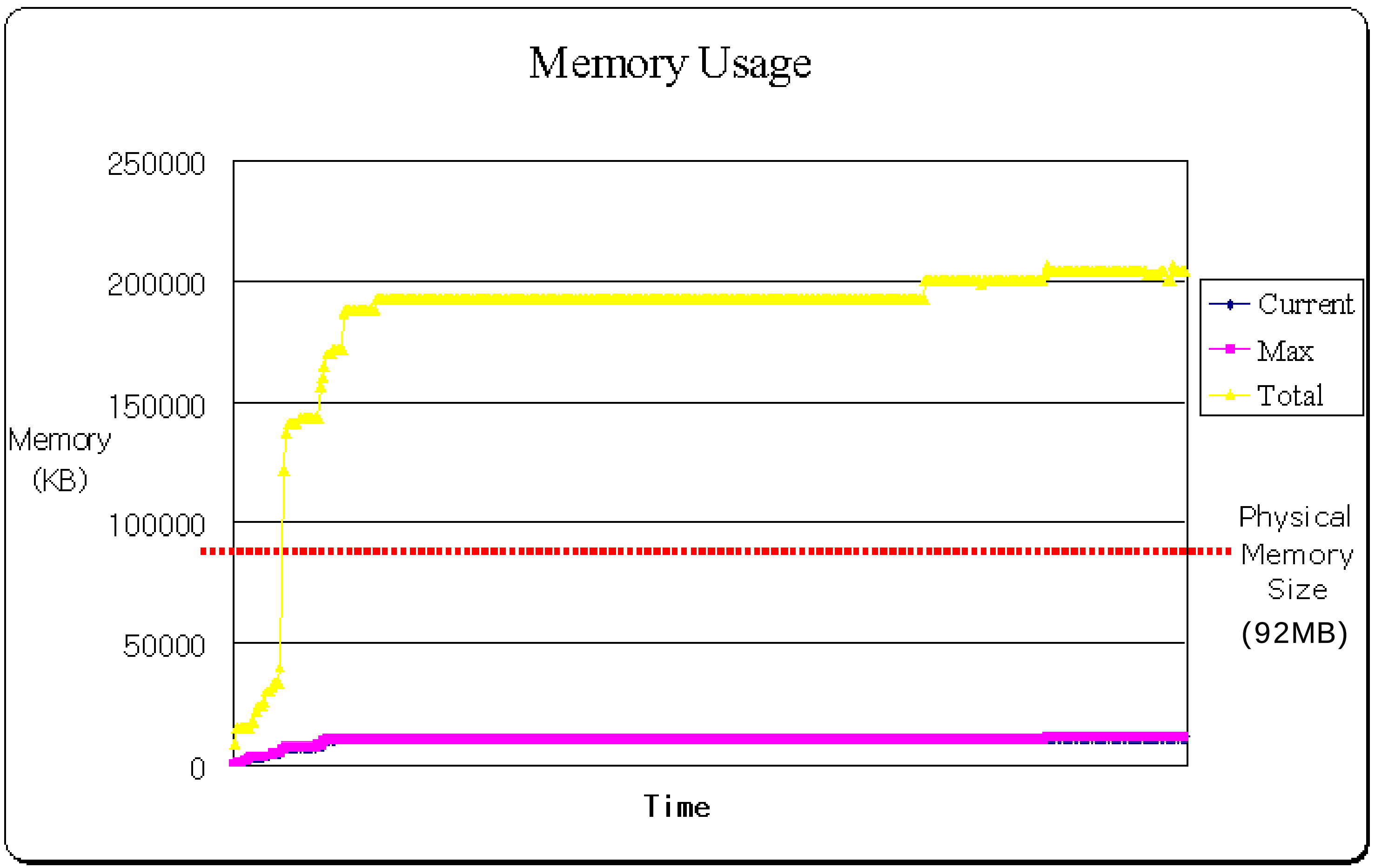
Overcommit

- The total aggregation of virtual address space can be larger than physical memory size.
- Application can allocate 'large' memory without considering physical memory size.



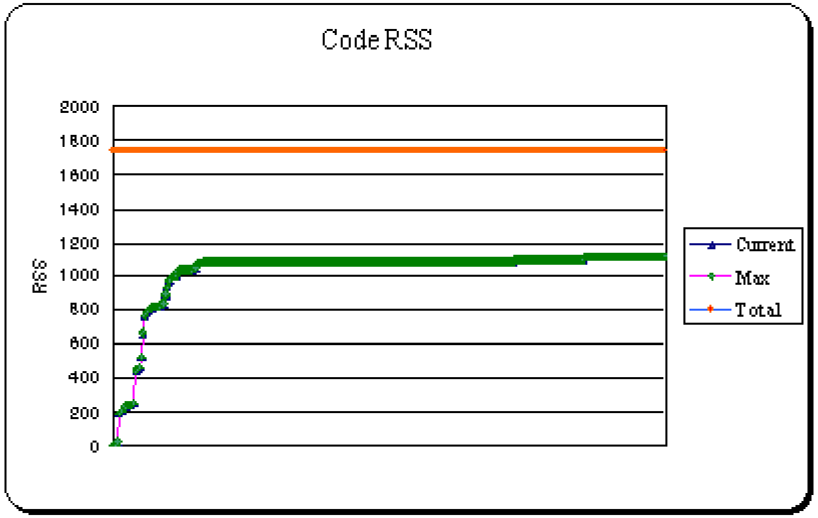
Address Space Usage: Virtual vs. Physical (1/2)

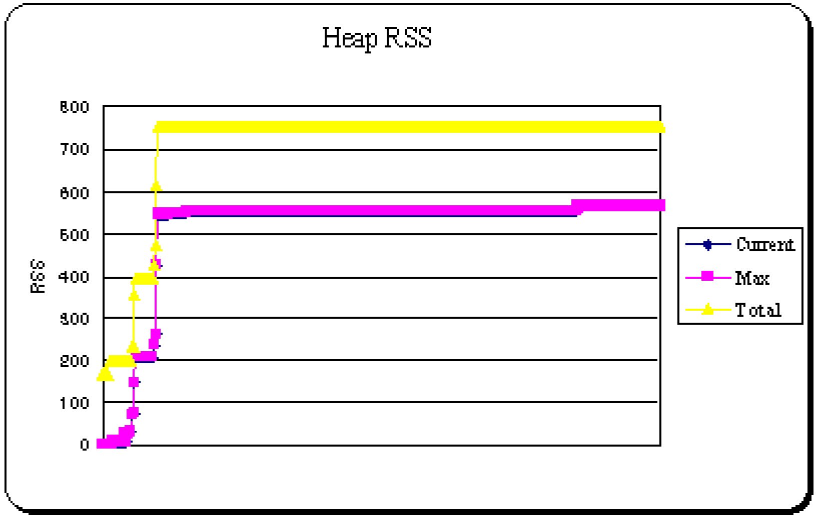
- Memory Usage Example.

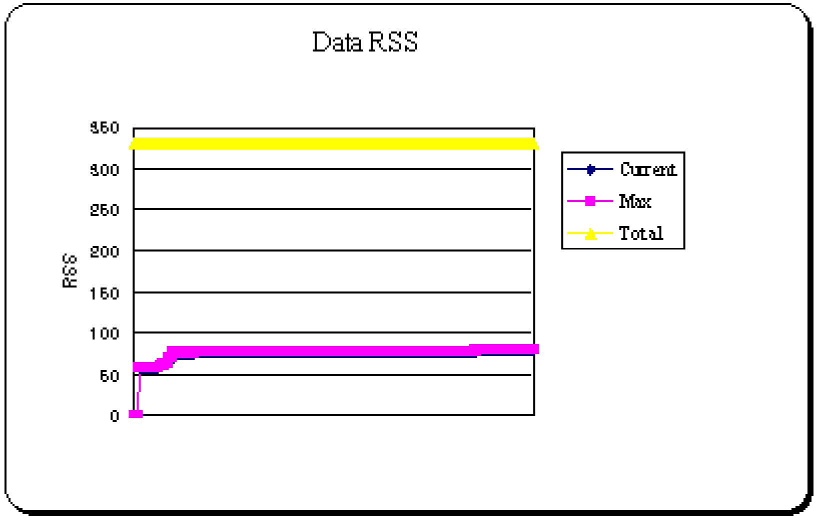


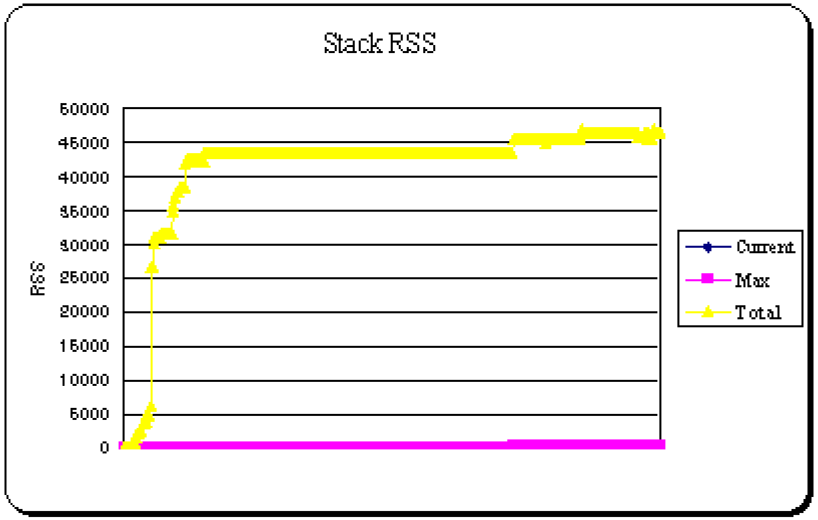
Address Space Usage: Virtual vs. Physical (2/2)

- Memory Usage Example. (per each area)







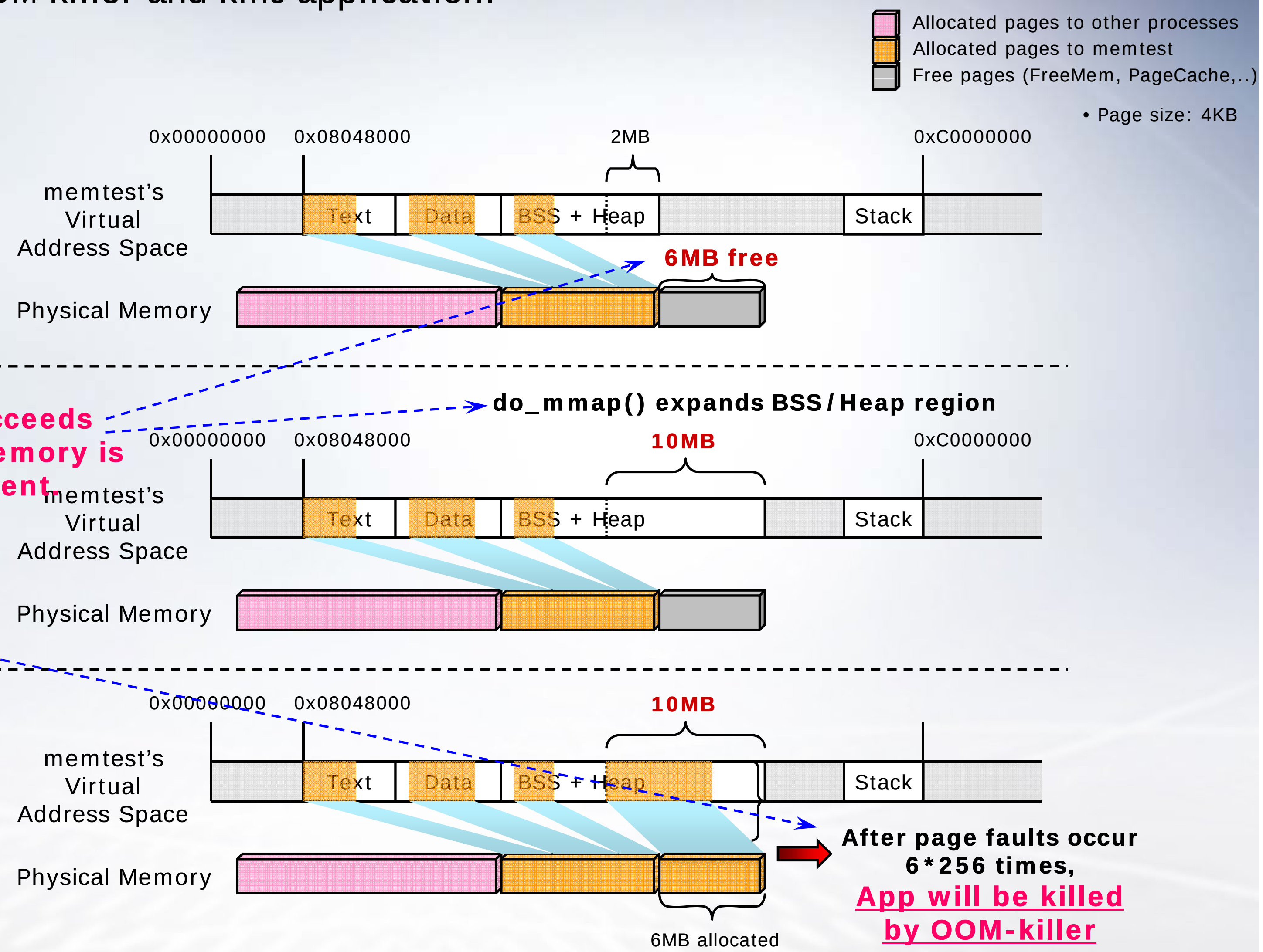


Overcommit Problem (1/2)

- Allocation request (ex. malloc()) larger than available memory size always succeeds.
- Linux invokes OOM killer and kills application.

```
main(void)
{
. . .

char *p;
p = malloc(10M);

. . .


memset(p,0,10M);

. . .
```

Malloc() succeeds even when memory is not sufficient.

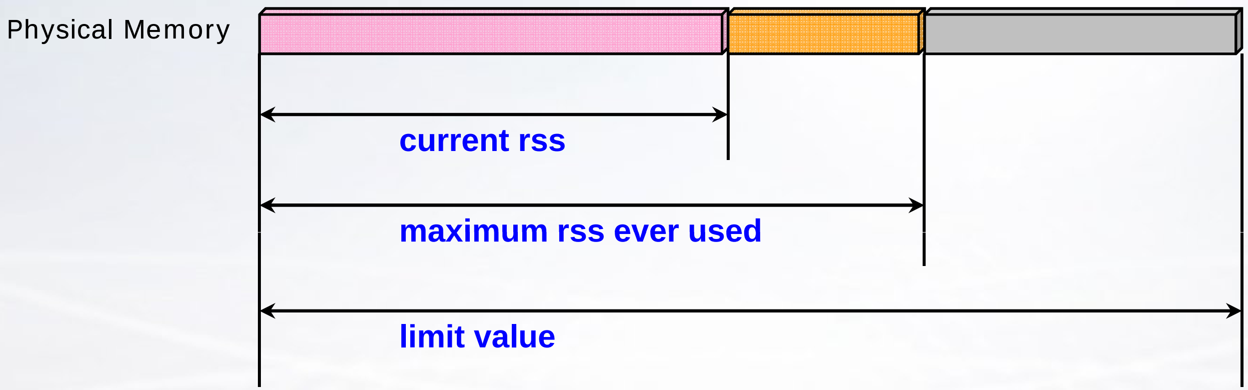


Overcommit Problem (2/2)

- Generally, SW developer thinks :
  - malloc() will return NULL, if the system has no available memory.
- Then, developers add error handling code when malloc() returns NULL.
- But, indeed linux kernel doesn't return NULL.
  - Instead, invokes OOM killer when allocated spaces are really used in low memory condition.
- So, the system gains no opportunity to handling 'allocation failure' error.
- It's a problem for all embedded linux systems.

Approach & Implementation (1/5)

- Limit maximum physical memory size per each application.
- If memory usage exceeds limitation, malloc() returns NULL.
- Data Structure
  - cur_rss : current rss (= current physical memory usage)
  - max_rss : maximum rss ever been used since process created
  - limit_max : rss limitation of each process can use

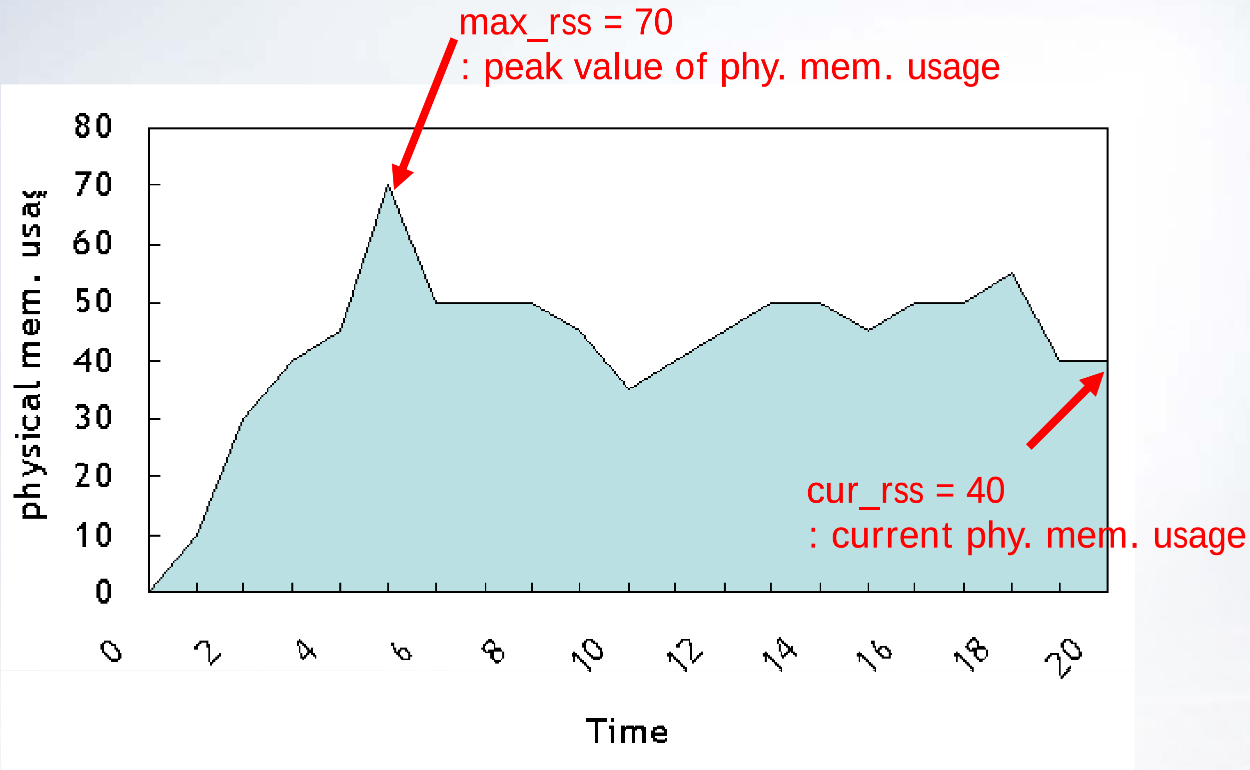


cur_rss ≤ max_rss

Approach & Implementation (2/5)

- Why 'max_rss' is needed?
  - max_rss : maximum rss ever been used since process created

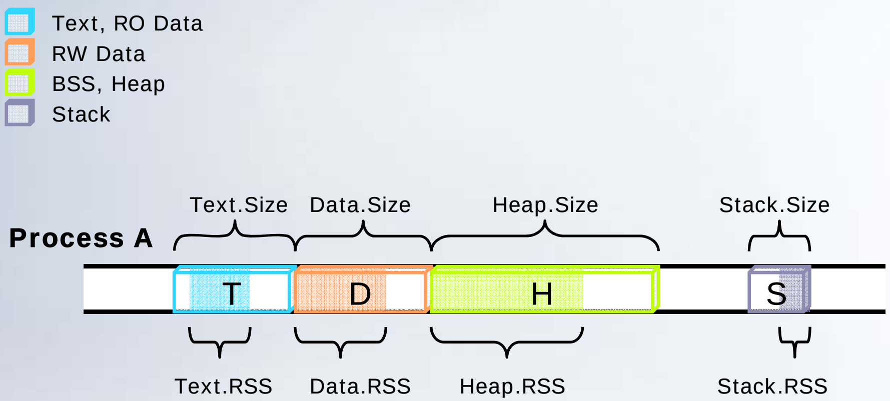
[example : trace physical memory usage]



We can say that this application's real memory usage is 70.
With the 'max_rss' value, we can limit application's memory usage.

Approach & Implementation (3/5)

- 2 phases : memory usage profiling, run-time allocation control
  - Profiling: collect max RSS for each address section (text, data, ..) of target process
  - Run-time memory allocation control: admission control of memory area allocation
- Based on profiling result, we can do allocation control.



- App's maximum memory usage

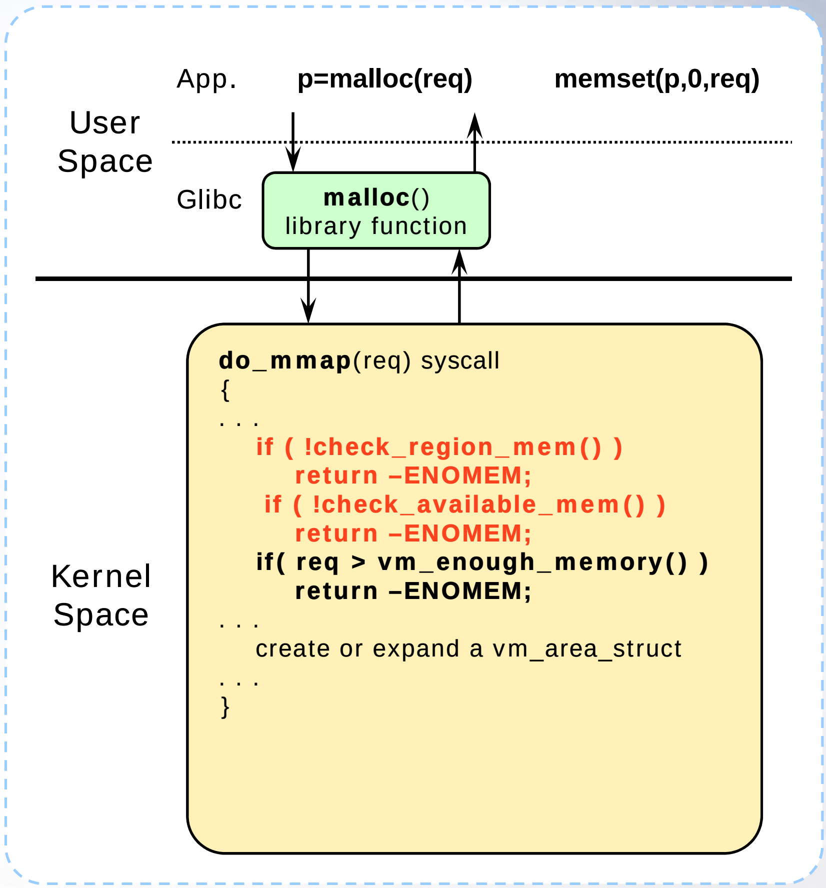
$$M^{app}_{required} = i.T.\max + i.D.\max + i.H.\max + i.S.\max$$

- App's total sum of virtual address space

$$M^{app}_{virt\_total} = i.T.size + i.D.size + i.H.size + i.S.size$$

$$M^{app}_{required} << M^{app}_{virt\_total}$$

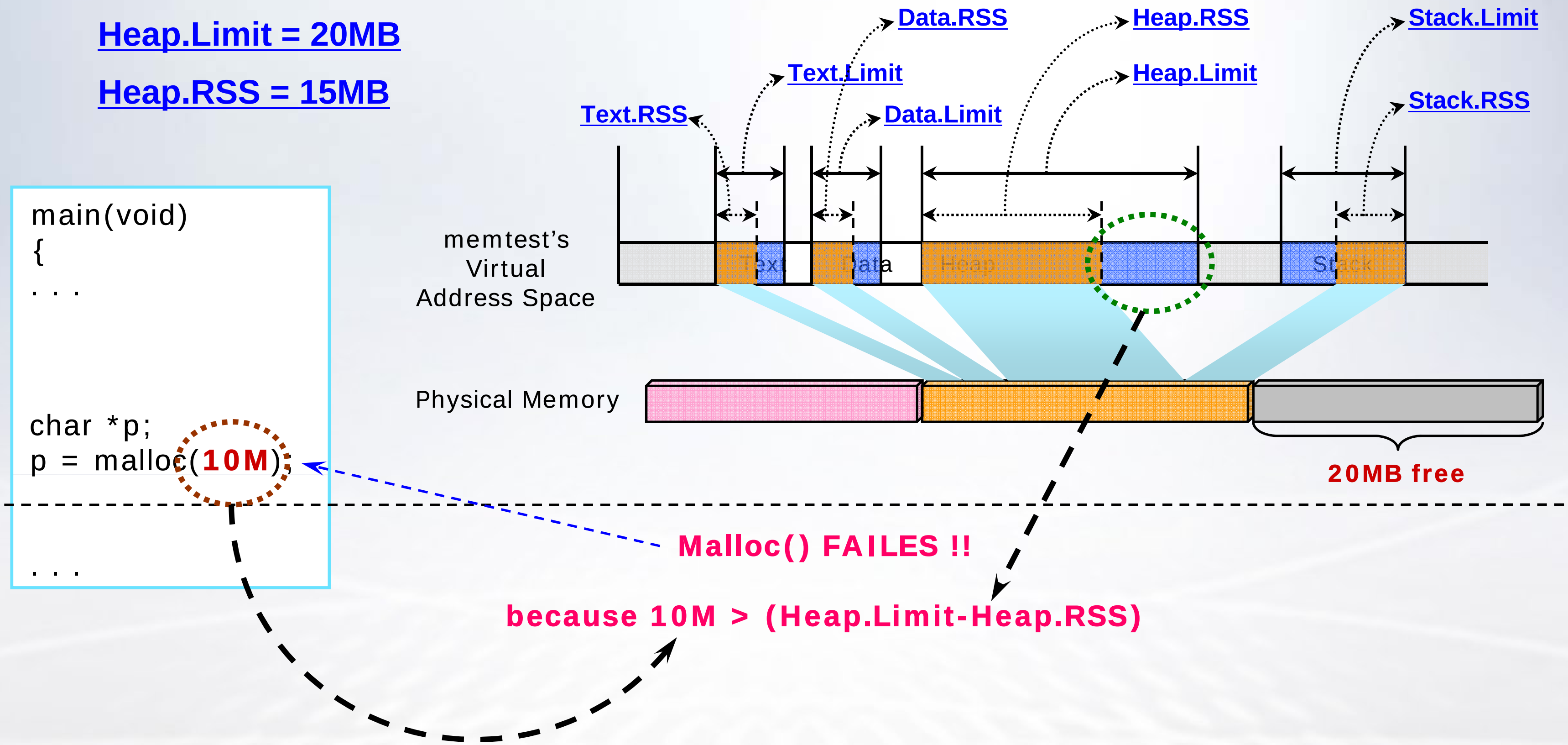
Memory Usage Profiling



Run-time Memory Allocation Control

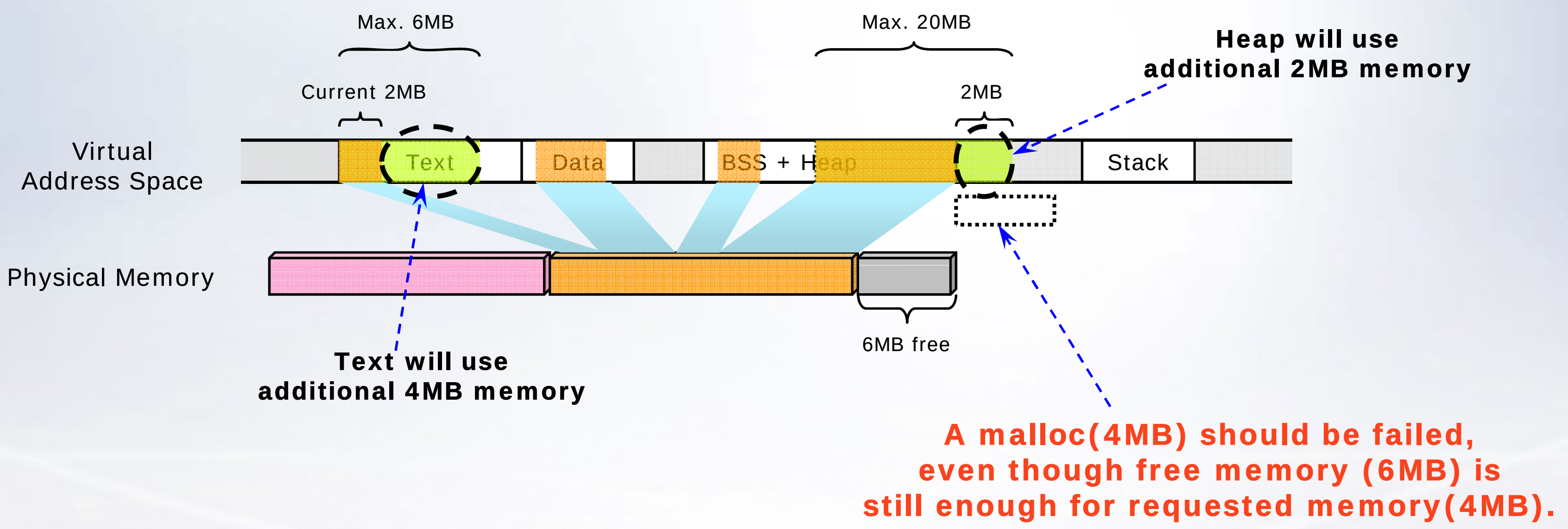
Approach & Implementation (4/5)

- With the limit value and current rss, we can decide allocation will succeed or not.
- To succeed in allcation, 2 conditions must be fulfilled.
  - **allocation size < limit - rss**
  - **allocation size < system free memory**
    - **Because another process can use free memory.**



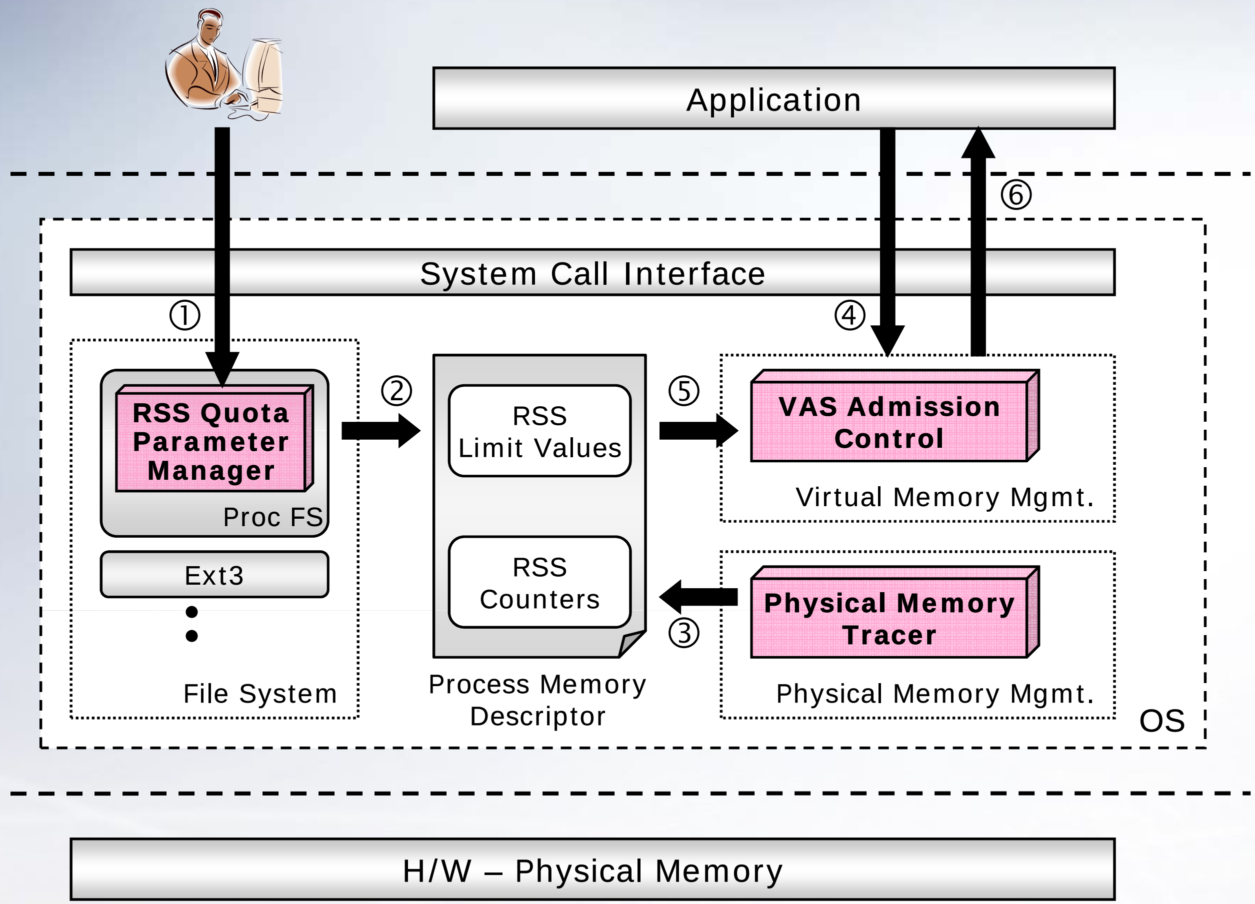
Approach & Implementation (5/5)

- Other regions (text, data, stack region) uses memory as well as heap.
- → Keep memory usage per each region.
- Need for allocation control of fork(), exec(), mremap(), …



RSS Quota (1/3)

- System Diagram



① Setting RSS Quota

② Save settings into memory descriptor

③ Trace & update physical memory page counters

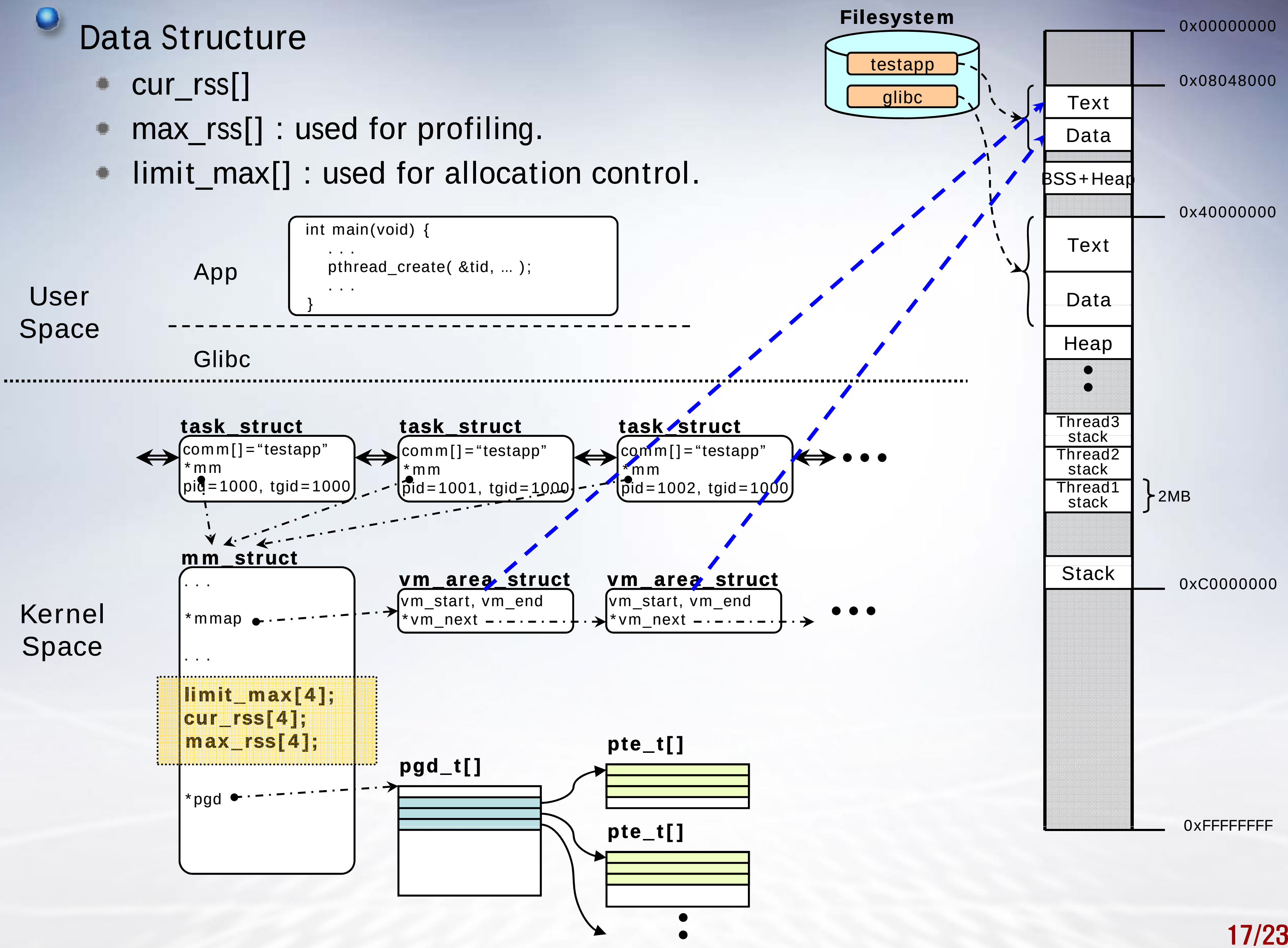
④ Memory allocation requests

⑤ Compare the counter with the limit value

⑥ Return results (success or fail)

RSS Quota (2/3)

- Data Structure
  - cur_rss[]
  - max_rss[] : used for profiling.
  - limit_max[] : used for allocation control.



RSS Quota (3/3)

- Results

```
# ./rssq_test 10
malloc start... : 1M x 10

malloc (000)
malloc (001)
malloc (002)
malloc (003)
malloc (004)
malloc (005)
malloc (006)
malloc (007)
malloc (008)
malloc (009)
malloc & memset completed

memory freeing...
memory free complete.

# _
```

test program
(1M x 10 allocation)

```
# ps
 PID  Uid     VmSize Stat Command
   1 0          528 S   init
   2 0              SW< [ksoftirqd/0]
. . .
 316 0          620 S   /bin/sh
 317 0          404 S   ./rssq_test 10
 318 0          664 R   ps

# cat /proc/317/rss
 2684     90     13      4
 Process Max :  10736K
    Code Max :    360K
    Data Max :     52K
   Stack Max :     16K
   Other Max :  10308K

# echo "2684 90 13 4 /test/rssq_test“
 > /proc/sys/vm/rss_quota

# cat /proc/sys/vm/rss_quota
 2684  90  13  4   /test/rssq_test
#
```

Restrict memory
based on profiling

```
# ./rssq_test 11
malloc start... : 1M x 11

malloc (000)
malloc (001)
malloc (002)
malloc (003)
malloc (004)
malloc (005)
malloc (006)
malloc (007)
malloc (008)
malloc (009)
malloc error
memory freeing...
memory free complete.

#
```

After restrict,
allocate 1M x 11

Limitation & Future Works

- Limitation
  - We must profile each application.
  - Inaccuracy of calculating system free memory.
- Future Works
  - Calculates free memory more accurately.
  - Improves PFRA (Page Frame Reclaiming Algorithm)
  - Shared Memory Accounting.

Another Approaches (1/3)

- No overcommit
  - It doesn’t use overcommit policy.
  - “echo 2 > /proc/sys/vm/overcommit_memory”
  - Pros
    - We can sure that OOM will never occur.
  - Cons
    - There’s no merit of demand paging.
    - Some or more applications may not run.

Another Approaches (2/3)

- OOM notify to application
  - When occurred lack of kernel memory, kernel notify it to user applications. And each application manages OOM situation.
  - WinCE
    - http://blogs.msdn.com/windowsmobile/archive/2006/08/16/702746.aspx
  - mem notify patch
    - http://lwn.net/Articles/267013/
  - Pros
    - Effective manipulation of OOM.
      - Because application knows well which memory allocation is useless than the other allocations.
  - Cons
    - Application developer should consider about OOM.
    - Also, the existing application code must be changed.

Another Approaches (3/3)

- Improves OOM policy
  - Improves victim selecting method.
  - Android platform
    - Android has its own victim selecting method.
    - There's "importance hierarchy" based on the state of components.
    - http://code.google.com/android/intro/lifecycle.html
  - Pros
    - Effective than kernel OOM killer's victim policy.
  - Cons
    - OOM still exists. Because it changes only victim policy.

Thank You.
Q&A