GenerOS: An Asymmetric Operating System Kernel for Multi-core Systems

Authors:
Qingbo Yuan, Jianbo Zhao, Mingyu Chen, Ninghui Sun
Institute of Computing Technology
Chinese Academy of Sciences
yuanbor@ncic.ac.cn

Speaker:
SuZhen Wu
Huazhong University of Science & Technology
Motivation

- Symmetric multithread operating system such as Linux suffers from lock contention and cache pollution

- **Lock contention**
  - As more cores are packaged into a single chip, there are too many cores in a system
  - Each core has the ability to trap into kernel
  - Too many procedures in kernel -> serious lock contention

- **Cache pollution**
  - Applications and kernel run on the same core
  - Applications may kick kernel’s cache line out of cache
  - And vice versa
Motivation

--- Lock Contention @ Linux

- **Contention Probability** = contentions / acquisitions
  - acquisitions: times acquiring lock
  - contentions: times encountering contention

- **Contention Efficiency** = hold time / (hold time + wait time)
  - hold time: time in critical region
  - wait time: time waiting for entering critical region
Motivation

---- Lock Contention @ Linux

![Contention Probability](image1)

![Contention Efficiency](image2)
Motivation
---- Cache Pollution @ Linux

![Bar Chart]

- Dcache Miss Ratio: 60%
- Icache Miss Ratio: 42%
- Dcache Lines Evicted: 96%
- Icache Lines Evicted: 92%
Motivation

Lock Contention
- More Cores
- More Contentions

Solution
- Decrease Cores in Kernel Mode

Cache Pollution
- Applications Run Together with OS

Solution
- Separate Kernel and Applications

GenerOS
Outline

1. Motivation
2. Architecture of GenerOS
3. Implementation of GenerOS
4. Evaluation of GenerOS vs Linux
5. Conclusion
In a symmetric multiprocessing system, Linux treats all cores as an equal which causes a lot of problems.

By contrast, GenerOS partitions processing cores into application core, kernel core and interrupt core:
- All of applications run on application core
- Their system calls are executed by kernel core
- Interrupts are all bound to interrupt core
Architecture

- Most of cores are used by applications
- A limited number of cores are used by kernel service
  - File System
  - Process
- Few number of cores are used to handle interrupt
Outline

1. Motivation
2. Architecture of GenerOS
3. Implementation of GenerOS
4. Evaluation of GenerOS vs Linux
5. Conclusion
Implementation

- GenerOS is developed based on Linux-2.6.25 @ x86_64 architecture
- In system call level, several kernel servers are developed
  - File system server (98 system calls)
    - `sys_open / sys_close / sys_read / sys_write`
  - Network server (15 system calls)
    - `sys_socket / sys_connect`
  - Signal server (12 system calls)
    - `sys_rt_sigaction`
  - IPC server (12 system calls)
    - `sys_msgget`
  - Process server (10 system calls)
    - `sys_fork`
  - Others (141 system calls)
    - `sys_brk`
int main(void)
{
    pid_t getpid();
    return 0;
}

pid_t generos_sys_getpid(void)
{
    req = generos_get_request();
generos_init_request(req);
    generos_send_to_kernel(req);
sleep();
    return pid;
}

while(generos_request_queue_is_not_empty(&process_queue)){
    req = generos_pick_request(&process_queue);
    switch(req->type){
    case GETPID:
        req->retvalue = sys_getpid();
        break;
    ......
    }
    wake_up_process(req->task);
}
It replaces the system call table of Linux

```c
const sys_call_ptr_t syscall_table[__NR_syscall_max+1] = {
    [__NR_read] = &generos_sys_read,
    [__NR_write] = &generos_sys_write,
    ......
    [__NR_timerfd_gettime] = &generos_sys_timerfd_gettime;
};
```

The left side keeps the same meaning with Linux which makes GenerOS compatible with Linux

The right side uses self defined function which will find a kernel core to handle its system call
Implementation

---- Kernel Core

- Two queues
  - Request queue
    - Receive system call requests from application core
  - Wait queue
    - Store the being handled system calls which are waiting for something

- One schedule method
  - Slim Schedule
    - Schedule system calls in this kernel core with almost zero overhead
Implementation

---- Binding Interrupt Handler

- Interrupt core is used to deal with most of interrupts from network interface, disk, or local timer
- In such way, both of application core and kernel core will have a clean execution environment
- GenerOS uses the method in Linux to bind interrupt handler to some processing core
Evaluation

---- Platform
Evaluation
---- Lock contention

<table>
<thead>
<tr>
<th>GenerOS</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00%</td>
<td>0.02%</td>
</tr>
<tr>
<td>0.04%</td>
<td>0.06%</td>
</tr>
<tr>
<td>0.08%</td>
<td>0.10%</td>
</tr>
<tr>
<td>0.12%</td>
<td>0.14%</td>
</tr>
<tr>
<td>0.16%</td>
<td>0.18%</td>
</tr>
</tbody>
</table>

Contention Probability

<table>
<thead>
<tr>
<th>Thread Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Contestion Efficiency

<table>
<thead>
<tr>
<th>Thread Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
Evaluation
---- Cache Pollution

- Dcache Miss Ratio
- Icache Miss Ratio
- Dcache Lines Evicted
- Icache Lines Evicted

<table>
<thead>
<tr>
<th>Kernel Proportion</th>
<th>linux</th>
<th>generos</th>
</tr>
</thead>
<tbody>
<tr>
<td>60%</td>
<td>44%</td>
<td>60%</td>
</tr>
<tr>
<td>42%</td>
<td>37%</td>
<td>42%</td>
</tr>
<tr>
<td>96%</td>
<td>92%</td>
<td>96%</td>
</tr>
<tr>
<td>92%</td>
<td>83%</td>
<td>92%</td>
</tr>
</tbody>
</table>
Evaluation
---- Single System Call
Evaluation
---- Single System Call

[Bar charts showing the comparison of system calls for 'open' and 'close' with 'read' and 'write'.]

- Kilobyte-cycles for 'open' and 'close':
  - 'l-enter': 30, 'l-handle': 20, 'l-exit': 10, 'flyin': 5, 'flyout': 15
  - 'g-enter': 25, 'g-flyin': 5, 'g-handle': 10, 'g-flyout': 10, 'g-exit': 5

- Million-cycles for 'read' and 'write':
  - 'l-enter': 20, 'l-handle': 10, 'l-exit': 5, 'flyin': 5, 'flyout': 5
  - 'g-enter': 15, 'g-flyin': 5, 'g-handle': 10, 'g-flyout': 5, 'g-exit': 5
Evaluation
---- TPC-H 1GB Power

The bigger the better
Evaluation
---- TPC-H 1GB Power

<table>
<thead>
<tr>
<th></th>
<th>application core</th>
<th>kernel core</th>
<th>interrupt core</th>
</tr>
</thead>
<tbody>
<tr>
<td>g-8000</td>
<td>0 ~ 14</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>g-8800</td>
<td>0<del>10, 12</del>14</td>
<td>11, 15</td>
<td>11, 15</td>
</tr>
<tr>
<td>g-8880</td>
<td>0<del>6, 8</del>10, 12~14</td>
<td>7, 11, 15</td>
<td>7, 11, 15</td>
</tr>
<tr>
<td>g-8888</td>
<td>0<del>2, 4</del>6, 8<del>10, 12</del>14</td>
<td>3, 7, 11, 15</td>
<td>3, 7, 11, 15</td>
</tr>
<tr>
<td>g-c000</td>
<td>0~13</td>
<td>14, 15</td>
<td>14, 15</td>
</tr>
<tr>
<td>g-e000</td>
<td>0~12</td>
<td>13, 14, 15</td>
<td>13, 14, 15</td>
</tr>
<tr>
<td>g-f000</td>
<td>0~11</td>
<td>12, 13, 14, 15</td>
<td>12, 13, 14, 15</td>
</tr>
</tbody>
</table>
Evaluation
---- Httpperf

![Bar chart showing replies per second vs. requests per second for different GenerOS versions and Linux.]

- The x-axis represents requests per second ranging from 200 to 1000.
- The y-axis represents replies per second, with the note that "the more the better."
Conclusion

- GenerOS is an asymmetric kernel which is designed to deal with the problems faced in traditional symmetric kernel.
- Being compatible with Linux, GenerOS does not need to modify, recompile, or relink applications, or libraries.
- Experiments with two typical workloads on 16-core AMD machine show that GenerOS behaves better than original Linux kernel when there are more processing cores.
  - 19.6% for TPC-H using oracle database management system.
  - 42.8% for httpperf using apache web server.
Thank you very much!

Any question?

Please contact the author yuanbor@ncic.ac.cn