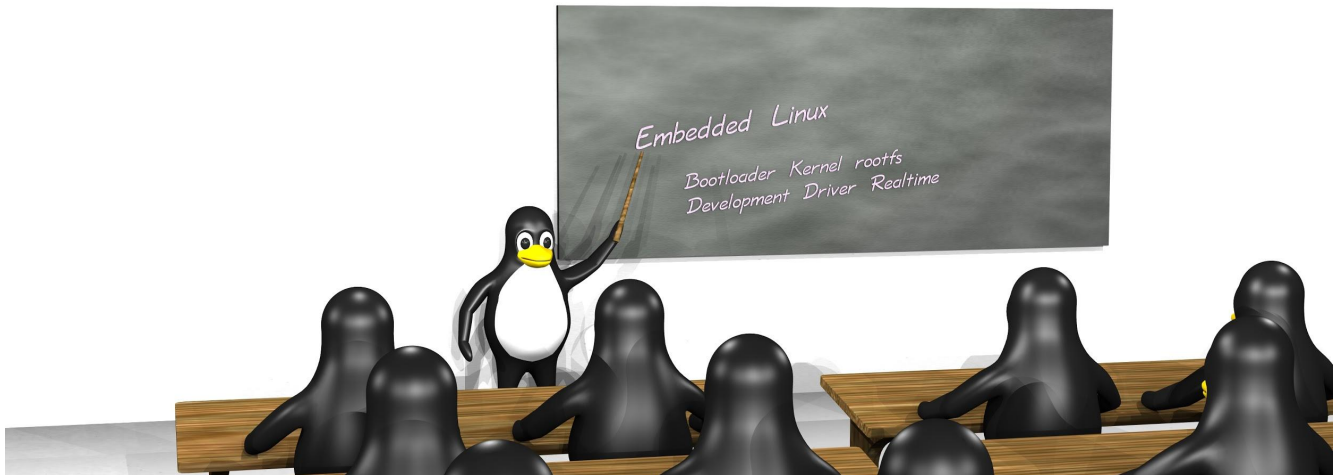
Graceful Failure During OOM Conditions

Ravi Mangar, Dylan Fox, Katrina Siegfried

Linux Memory Allocation

- Linux intends for “good” malloc() calls never to fail
- If a mmap/malloc call fails due to lack of memory, OOM killer is activated
- The OOM killer tries to kill processes to free memory



The OOM Killer

- Complicated score to decide which process is killed, but primarily based on size of memory usage
- Adjusted OOM Score in proc fs allows user to make a process less likely (or impossible) to be killed by OOM killer
- Can be set to -1000 which makes a process “unkillable” by the OOM Killer.
- But what if all killable processes have been killed?
- Deadlock, unresponsiveness, and system crash!

OOM Conditions Causes

- Coding errors are the most common cause of memory leaks
- Overcommit may cause OOM
- Fundamental mismatch between hardware and task

These events are *rare*, but consequences can be catastrophic

Target Systems

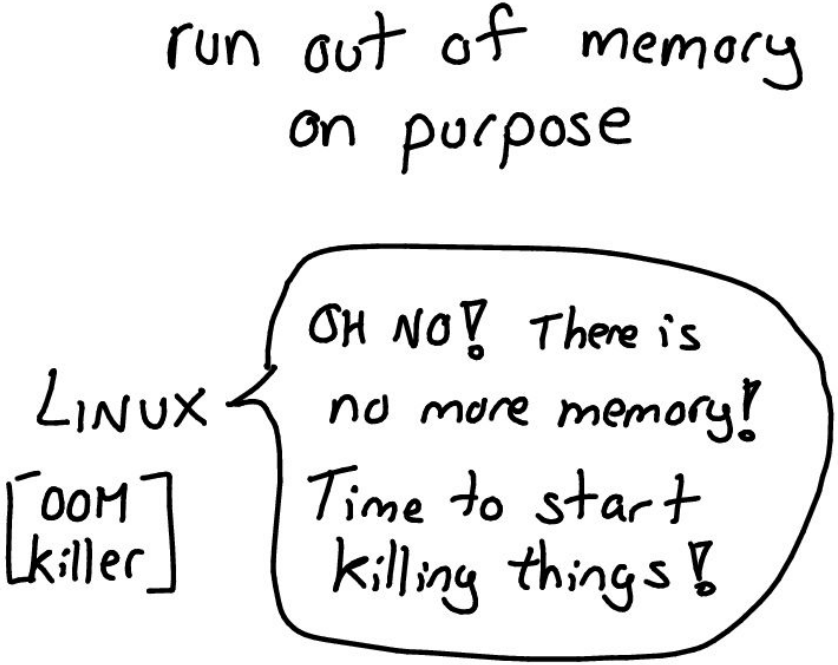
- Embedded systems or safety critical systems. Ex:
  - Medical Devices -- shut down and notify user of failure.
  - Self Driving Car Radar -- gently apply brakes and notify user to take over
- Processes that would lose data if killed. Ex:
  - Transmit data from a remote sensor.
  - Training neural networks: save network before shutting down.
- Logging. Ex:
  - Save extra information to debug out of memory conditions.
- Can expose APIs while maintaining device safety

Design Fulfillment

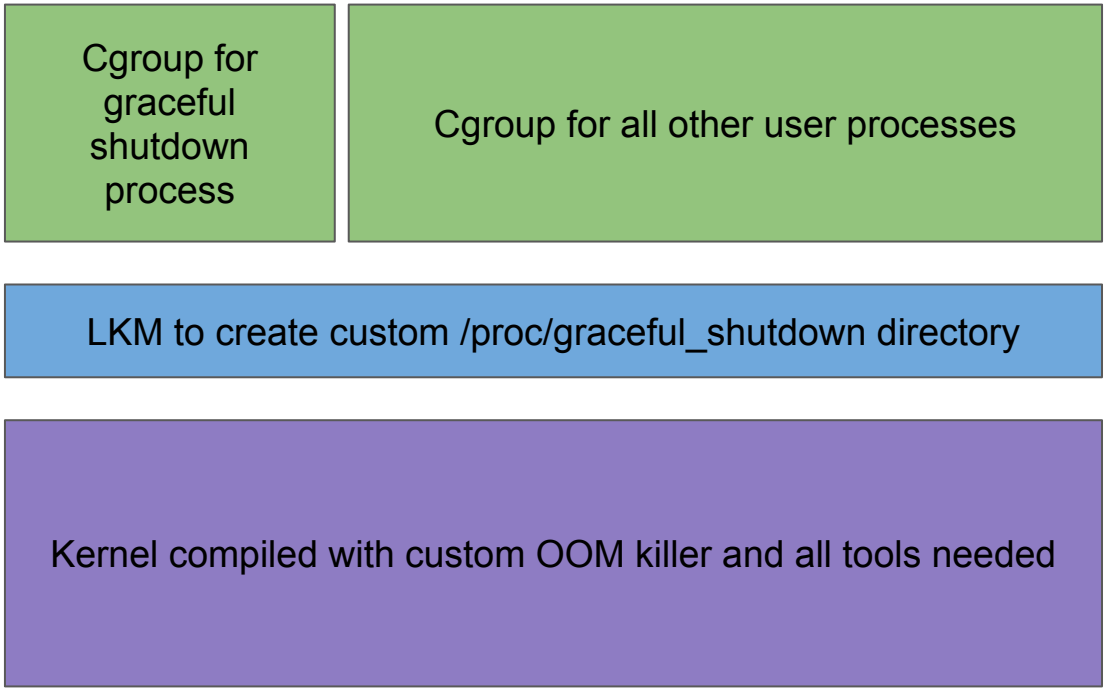
1. Graceful shutdown processes need to run reliably. ✓
2. Graceful shutdown processes should not trigger additional OOM conditions or be killed. ✓
3. The user should have a way to mark a running process as having a graceful shutdown procedure. ✓
4. Graceful shutdown processes should not run in kernel mode. These processes are written by an end user, so should run in user space. ✓

Not limited to embedded devices, will work with any 5.4 kernel dist.

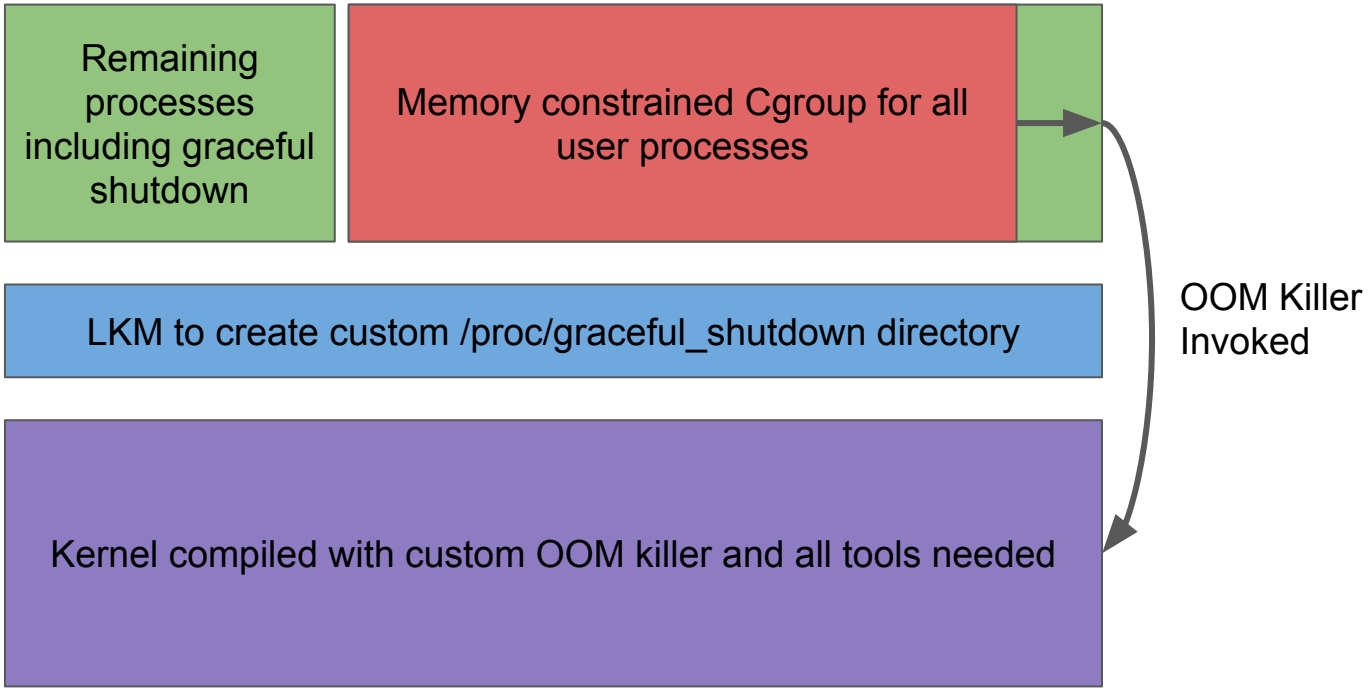
High-Level Overview of Functionality



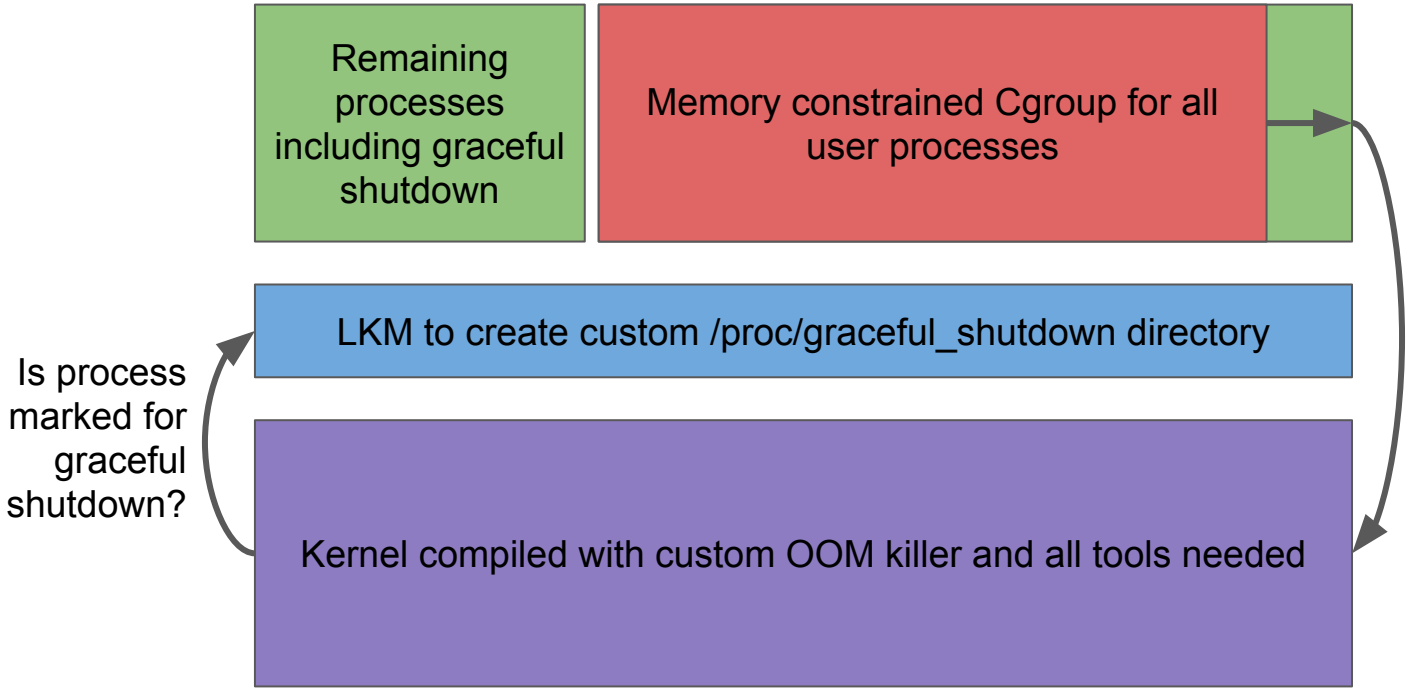
Graceful Shutdown



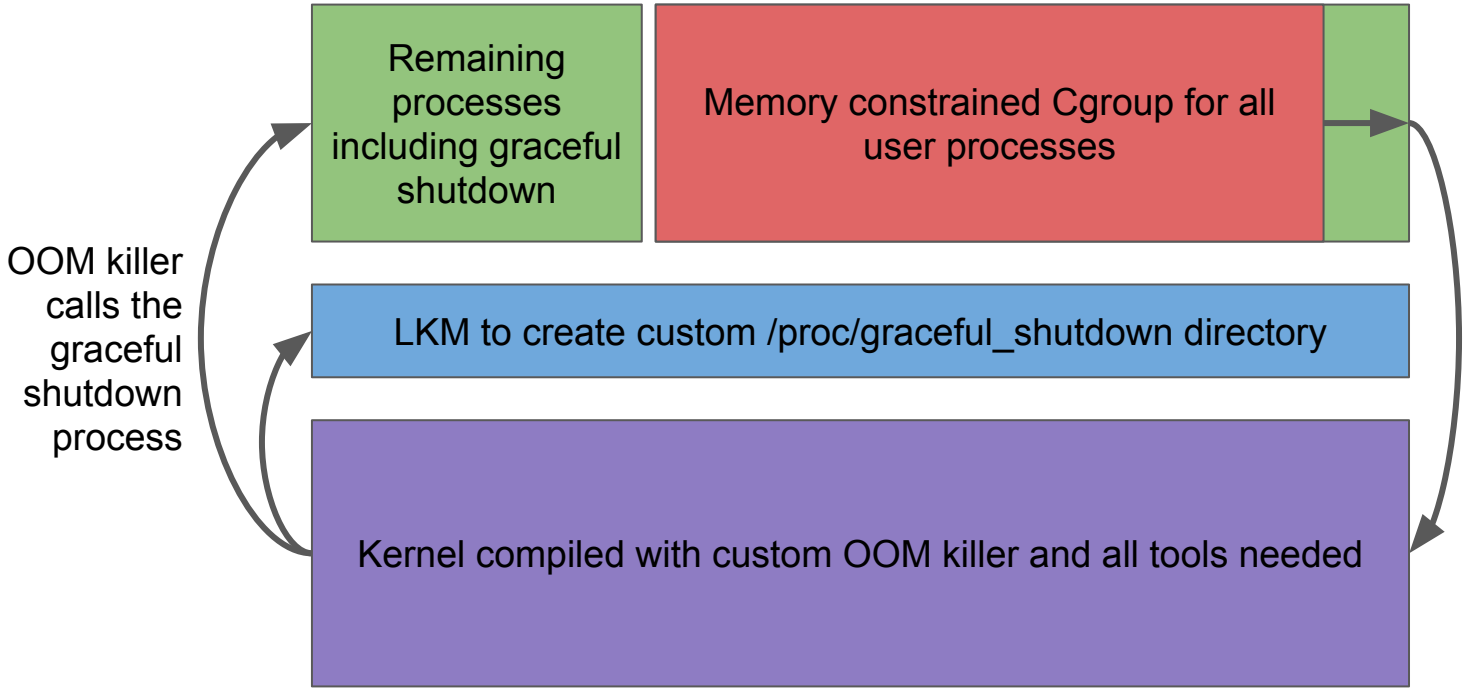
Step 1: OOM Killer Invoked



Step 2: OOM Killer queries /proc/graceful_shutdown



Step 3: Graceful shutdown process called



Implementation

1. Modify mm/oom_kill.c
   a. Tweak `select_bad_process()` to test if process has been marked for graceful shutdown.
   b. If process has been marked for graceful shutdown, execute user space program to handle shutdown.
2. Create file in `/procfs`
   a. Contains pid of process
   b. & path of user space executable

PID	PATH
1234	/usr/src/executable_1
1235	/usr/src/executable_2

System Configuration Changes

- Set Loadable Kernel Module to be inserted at boot.
- Enable memory management in GRUB
- Create cgroup rules

Invoking user process from kernel

`select_bad_process()`

Used by `oom_evaluate_task()`

`oom_kill.c`

```
pid = (int)oc->chosen->pid;
printk(KERN_ALERT"OOM pid choosen, pid is: %d", pid);
gs_path = get_graceful_shutdown_path(pid);
printk(KERN_ALERT"gs_path for %d is: %s", pid, gs_path);
if (gs_path != NULL)
        argv[0] = gs_path;
        return_val = call_usermodehelper(argv[0], argv, NULL, UMH_WAIT_PROC);
        printk(KERN_ALERT"return val from usermodehelper is: %d", return_val);
        kfree(gs_path);
```

/proc/graceful_shutdown

Create entry in `/procfs`

Read relevant directory

(Ideal Soln; append to proc file)

proc_create_module. c

```
#define PROCFS_NAME "graceful_shutdown"

graceful_shutdown_file = proc_create(PROCFS_NAME,0666,NULL,&proc_fops);
```

```
kernel_read_file_from_path(directory, &buffer, &size, max_size, READING_MODULE);

char *file_contents = (char *)buffer;

filp_close(f, NULL);

return file_contents;
```

Testing: Creating an OOM condition

oom_condition_creator.c

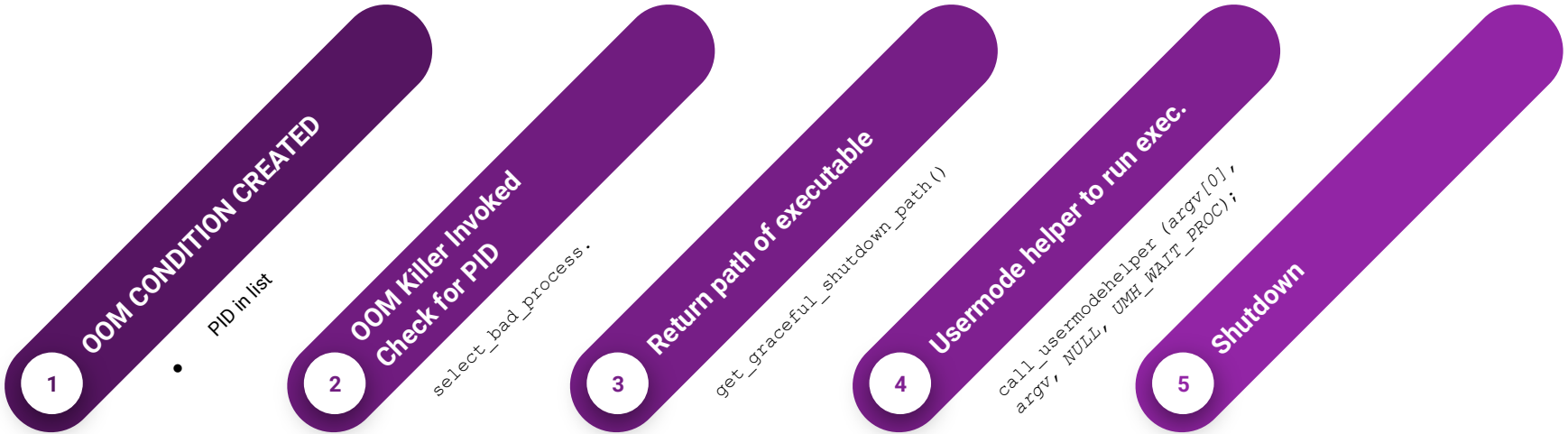
```
void* huge_leak = malloc(SIZE);

memset(huge_leak, '0', SIZE);

printf("%lu bytes lost so far!\n", i * SIZE);

i += 1;
```

Flow


1
OOM CONDITION CREATED
PID in list
2
OOM Killer Invoked
Check for PID
select_bad_process.
3
Return path of executable
get_graceful_shutdown_path()
4
Usermode helper to run exec.
call_usermodehelper (argv[0],
argv, NULL, UMH_WAIT_PROC);
5
Shutdown

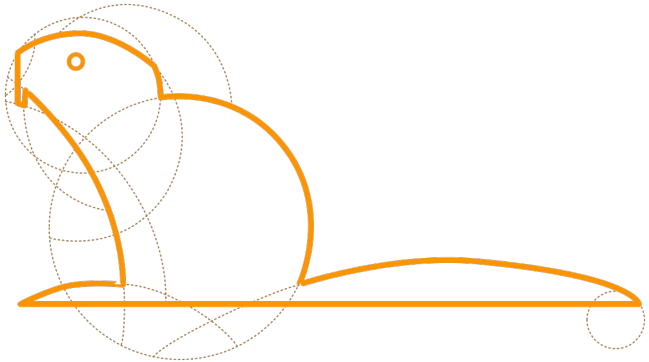
Installation and Usage Overview

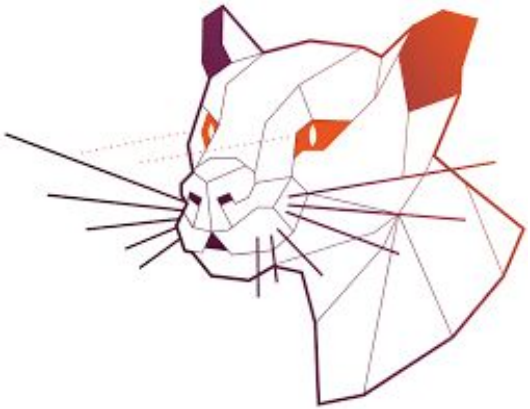
Installation Preconditions

OS: Ubuntu 18.04 (4.15 kernel) or greater

Network connectivity (get mirrors)

Sudoer capabilities





Set up files to compile kernel for new VM

1. Get mirror for 5.4.1 kernel

2. Install all libraries needed for compile

3. Copy custom config with needed flags

setup_new_vm.sh

```
#!/bin/bash

LINUX_SRC_DIR='/usr/src/linux-5.4.1'

# remove old log files
rm -f sys_install.log install.log

# get kernel source in background while installing stuff.
(wget https://mirrors.edge.kernel.org/pub/linux/kernel/v5.x/linux-5.4.1.tar.gz \
&& sudo tar -xvf linux-5.4.1.tar.gz -C/usr/src) > install.log 2>&1 &

#packages for kernel compilation and sshing into vm.
(sudo apt-get install gcc make libncurses5-dev bison flex libssl-dev libelf-dev openssh
sudo apt-get update -y &&
sudo apt-get upgrade -y &&
sudo systemctl enable ssh) > sys_install.log 2>&1 &

# wait for both to finish, then setup config.
echo "Waiting for both jobs to finish!"
wait $(jobs -p)
cd "$LINUX_SRC_DIR" && sudo make defconfig
cd - && sudo cp kernel.config $LINUX_SRC_DIR/.config
```

Create custom kernel and all necessary files

4. “Inject” custom OOM code

5. Build kernel

6. Build LKM

7. Build Test Programs

8. Reboot into kernel

Makefile

```
LINUX_SOURCE_DIR:=/usr/src/linux-5.4.1/

.PHONY: all proc tests

all: proc tests linux_built.txt

linux_built.txt: oom_kill.c
	/bin/bash -c "sudo ./copy_oom_kill_file_and_build.sh $(LINUX_SOURCE_DIR)"
	rm -f "linux_built.txt"
	touch "linux_built.txt"

tests:
	$(MAKE) -C test_programs

proc:
	$(MAKE) -C proc_filesystem
	$(MAKE) install -C proc_filesystem
```

Setup Running Environment

1. Set LKM for proc module to run at boot

2. Set up Cgroup to limit memory

3. Create systemd file to run Cgroup at boot

4. Apply Cgroup rules and restart systemctl

setup_initial_running_env.sh

```
14  # add the lkm for the proc directory to run at boot every time
15  echo "Setting /proc/graceful_shutdown to initialize on every boot....." &&
16  echo 'proc_create_module ' >> /etc/modules &&
17
18  # enable the memory management if not already enabled
19  echo "Setting up graceful_shutdown cgroups....." &&
20  echo '# Uncomment to enable memory management' >> /etc/default/grub &&
21  echo 'GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1"' >> /etc/default/grub &&
22  sudo update-grub &&
23  sudo touch /etc/default/grub.d/mem_cgroup
24  echo cgroup/cgroup_grub >> /etc/default/grub.d/mem_group
25
26  #use template for global config to create a global config
27  sudo cp /usr/share/doc/cgroup-tools/examples/cgred.conf /etc/ &&
28
29  #create file to specify groups
30  sudo touch /etc/cgconfig.conf &&
31  cat cgroup/cgroup_cgconfig.conf >> /etc/cgconfig.conf &&
    --
                                    ⋮
41  # create the systemd files for config and rules
42  echo "Setting up cgroups to start at boot....." &&
43  sudo touch /etc/systemd/system/cgconfigparser.service &&
44  cat cgroup/cgroup_systemd_config >> /etc/systemd/system/cgconfigparser.service &&
45  sudo touch /etc/systemd/system/cgrulesparser.service &&
46  cat cgroup/cgroup_systemd_rules >> /etc/systemd/system/cgrulesengd.service &&
47
48  # restart systemctl to apply the changes above
49  sudo systemctl daemon-reload &&
50  sudo systemctl enable cgconfigparser --now &&
51  sudo systemctl enable cgrulesengd --now
52
```

Usage

1. Pass process PID and graceful shutdown path to LKM or edit file directly using the following format

```
echo "<pid> <file path> << graceful_shutdown_list.txt
```

2. Execute user processes

```
cgexec -g memory:memlimit <program> <args>
```

Enhancements under way

Completion of full setup automation

Enhanced error checking

Benchmarking

Future Work

More predictive algorithms for OOM

Implementation / demonstration across different embedded devices

Additional support for other kernel versions and OS distributions

Testing

- Created OOM with large memory leaks.
- Can run as the process with the graceful shutdown process, or another non-killable process.

oom_condition_creator.c

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE 1000000000

int main() {
  printf("PID is: %d", getpid());
  fflush(stdout);
  long unsigned int i = 1;
  while(1) {
    sleep(1);
    void* huge_leak = malloc(SIZE);
    memset(huge_leak, '0', SIZE);
    printf("%lu bytes lost so far!\n", i * SIZE);
    i += 1;
  }
  return 0;
}
```

Questions?

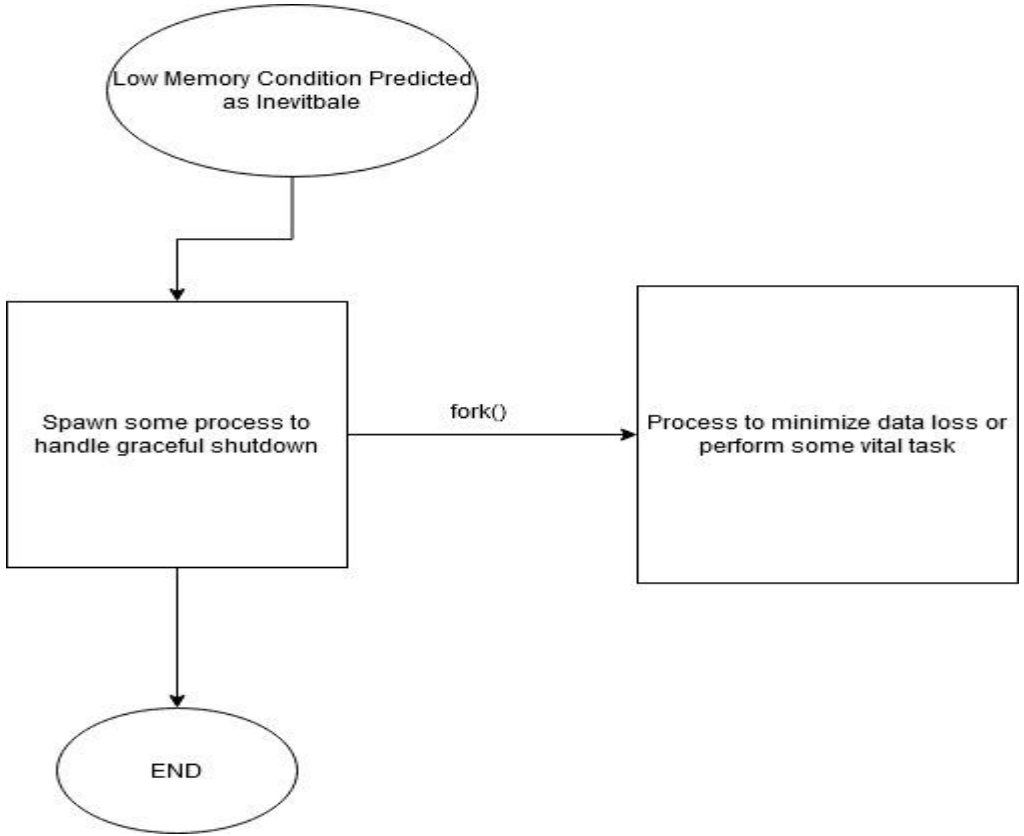
https://github.com/siegfrkn/csci5573-project

The following are extra or obsoleted slides

Handling OOM Conditions -- Graceful failure

- Currently two options -- let OOM killer kill process or mark process as unkillable and hope system can kill other processes
- In some situations , it may be better for a process to fail and execute a “graceful shutdown” procedure rather than get killed by OOM killer, or cause an unresponsive state.
- Our project adds the ability to run a user space process before killing a process in response to an OOM condition.

Spawning Processes

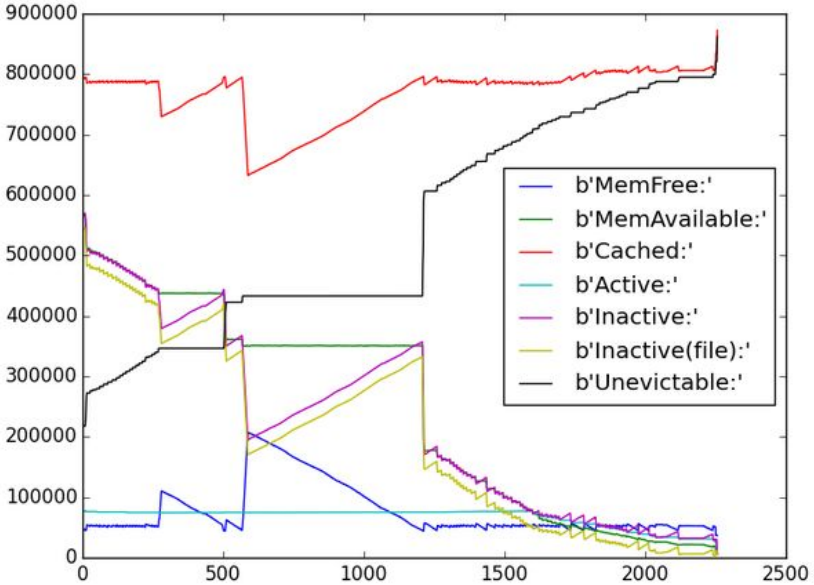


Spawning Processes cont’d

System Function	Process to Spawn
Data Logging	Write data stream to file
Safety Critical	Barebone Functionality
Monitoring Equipment	Raise Alarm

Predicting Out of Memory Condition

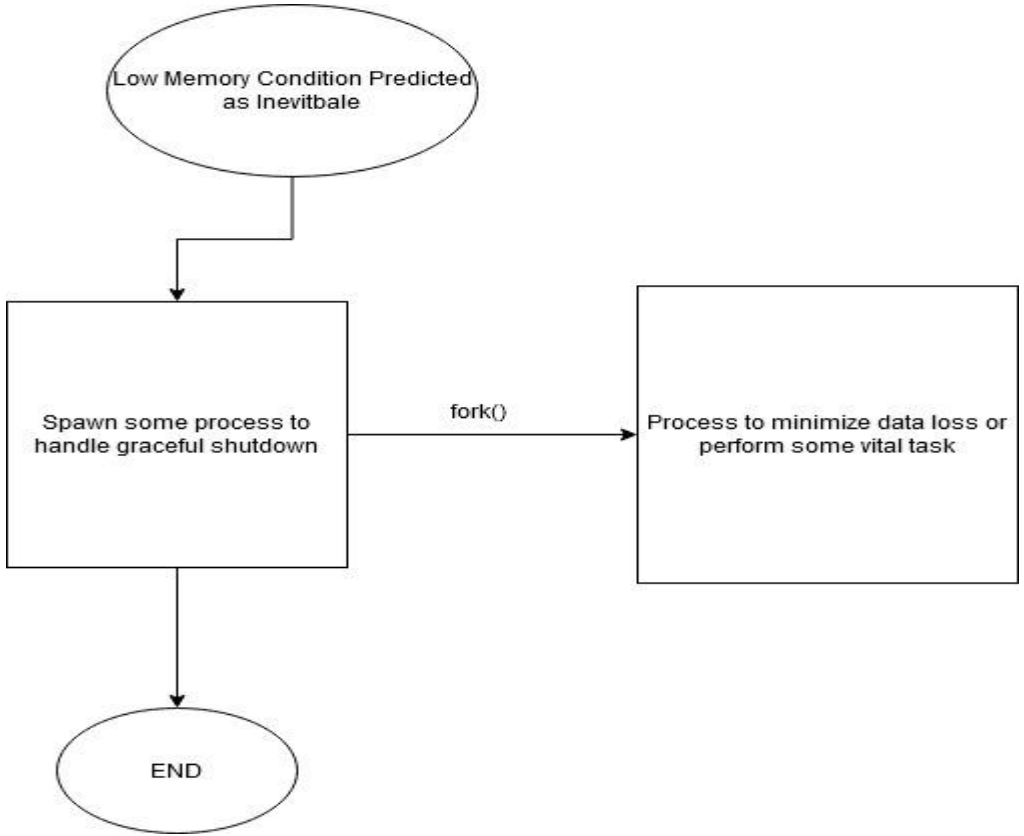
- Gather Statistics
  - OOM Scores
  - CPU Load
  - /proc/meminfo
- Combination of Factors
- Kickstart Pipeline
  - Minimise Data Loss
    - Or Barebone Functionality
  - Graceful Shutdown



Handling OOM Conditions -- Graceful failure

- Currently two options -- let OOM killer kill process or mark process as unkillable and hope system can kill other processes
- In some situations , it may be better for a process to fail and execute a “graceful shutdown” procedure rather than get killed by OOM killer, or cause an unresponsive state.
- Our project adds the ability to run a user space process before killing a process in response to an OOM condition.

Spawning Processes

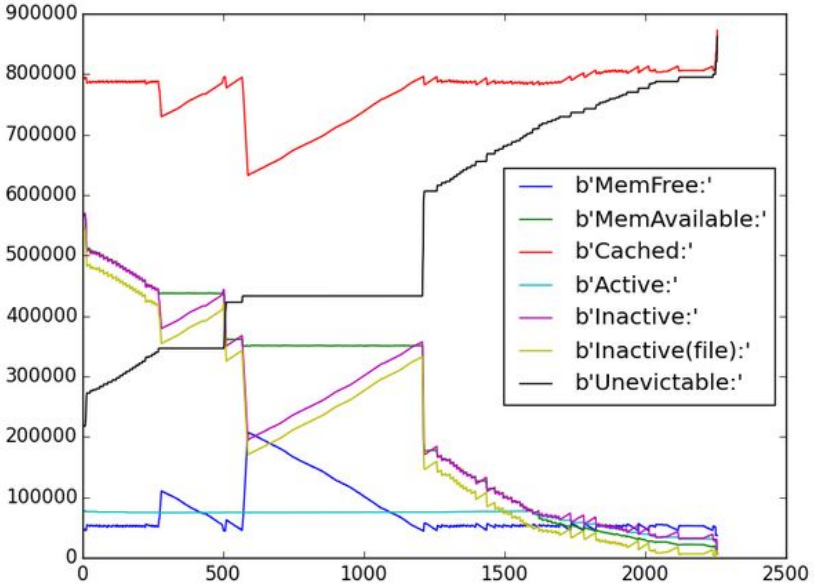


Spawning Processes cont’d

System Function	Process to Spawn
Data Logging	Write data stream to file
Safety Critical	Barebone Functionality
Monitoring Equipment	Raise Alarm

Predicting Out of Memory Condition

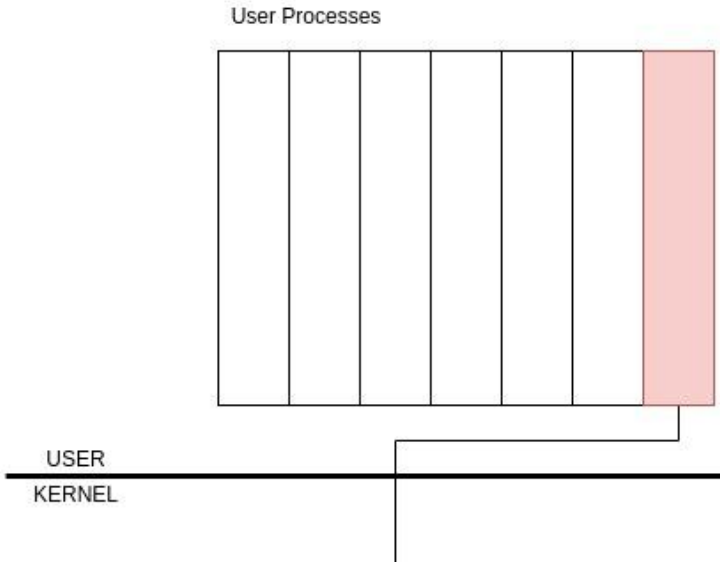
- Gather Statistics
  - OOM Scores
  - CPU Load
  - /proc/meminfo
- Combination of Factors
- Kickstart Pipeline
  - Minimise Data Loss
    - Or Barebone Functionality
  - Graceful Shutdown



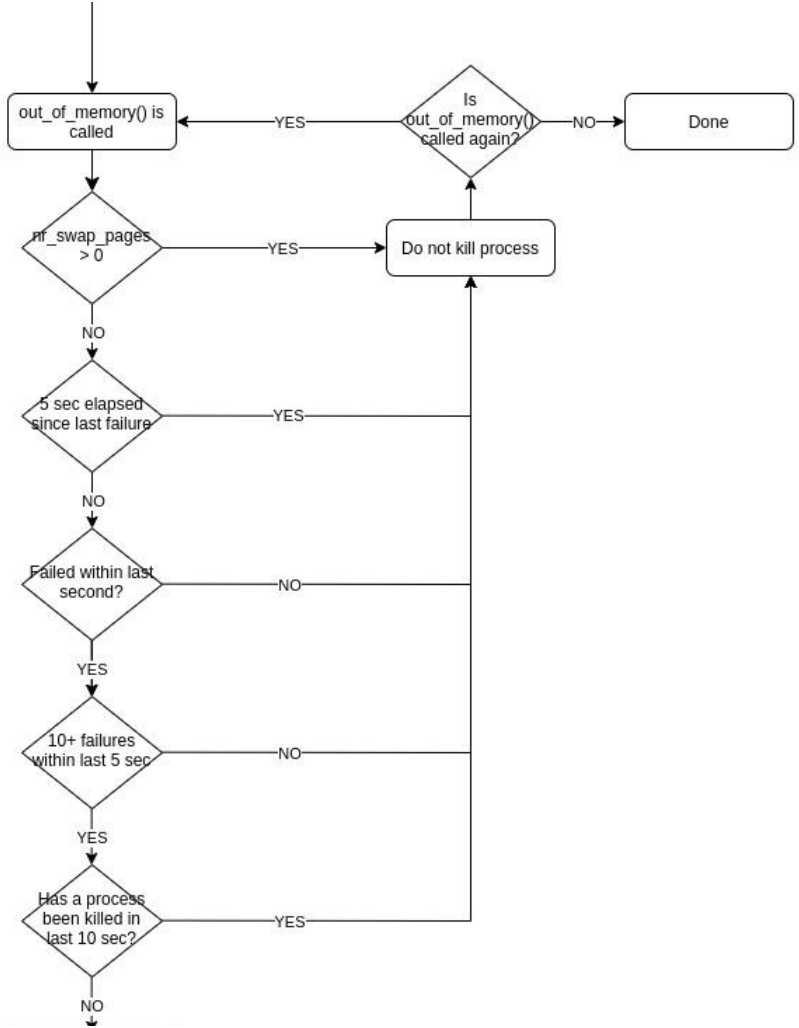
Demo

delayed

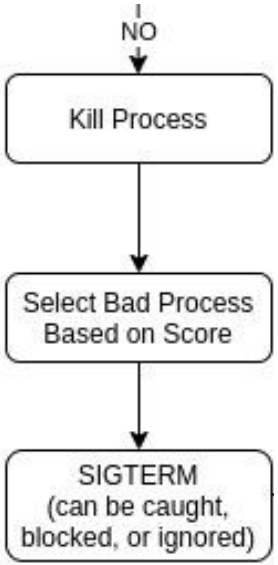
Graceful Shutdown



Graceful Shutdown

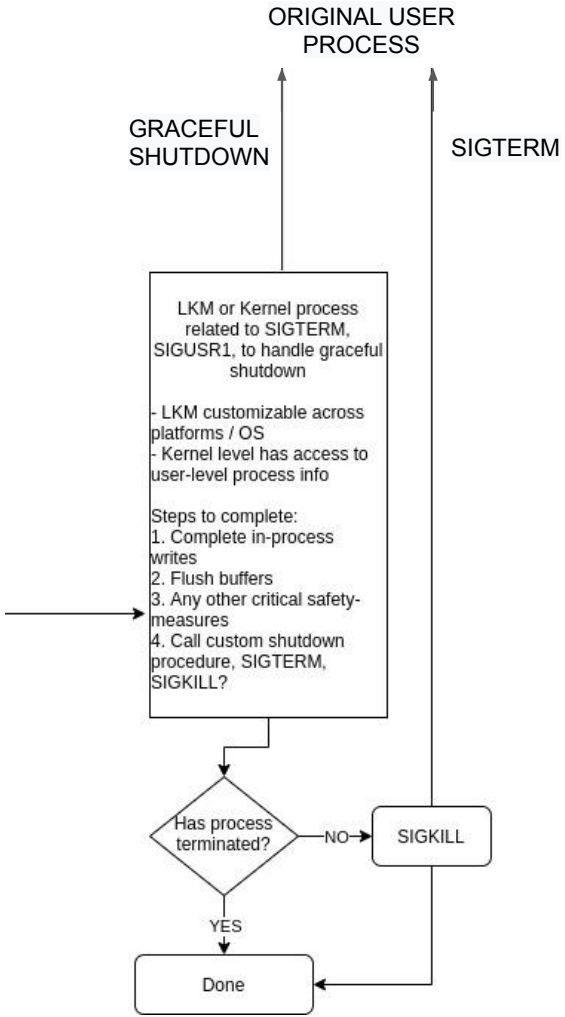


Graceful Shutdown



Graceful Shutdown

Graceful Shutdown



Graceful Shutdown - Method Pro / Con

LKM / System Call Intercept

PRO

- Wraps existing functionality
- Speed (as compared to process)
- Can still invoke kernel process if needed
- Users do not need to rebuild kernel

CON

- Complexity of newer kernel security
- Memory fragmentation
- Loaded late in boot cycle

Kernel Mode Process

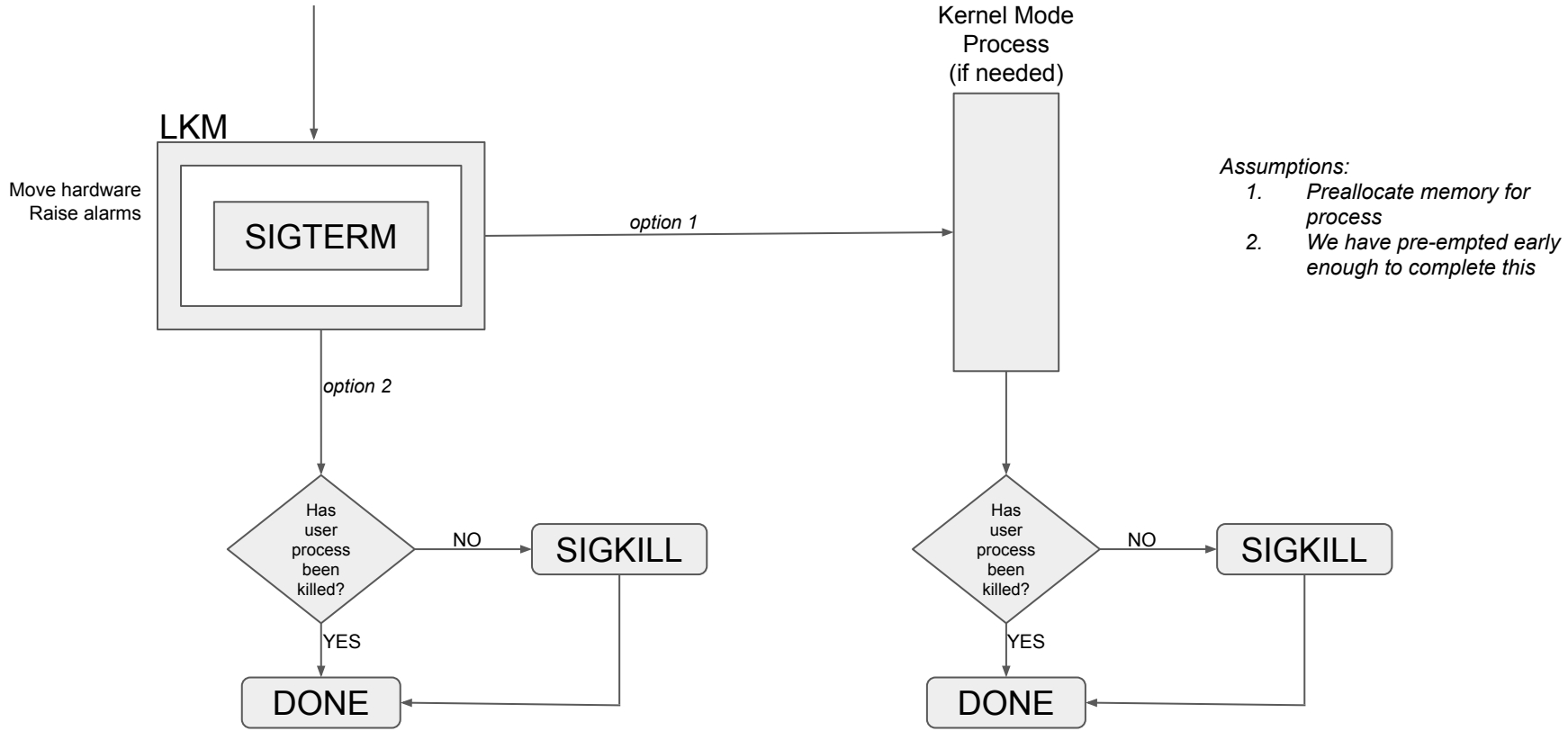
PRO

- Premarked as “unkillable”
- Access to kernel level and user space info

CON

- Slower to create / manage
- Likely slower than kernel execution

Graceful Shutdown - Potential Implementations



Assumptions:

1. *Preallocate memory for process*
2. *We have pre-empted early enough to complete this*

Embedded Systems

- Combo of hardware and software with specific purpose
- May be very simple or relatively complex
- Some run variants of linux
- Examples: remote sensors, medical devices, self driving car radar or gps, elevator control units, etc..

Memory, OOM, and Embedded Systems

- OOM conditions are less likely on embedded systems as workload is relatively well known and consistent. However…
- Overcommit may cause OOM
- Coding errors may cause memory leaks
- Fundamental mismatch between hardware and task