TECHNISCHE UNIVERSITÄT DRESDEN

SOFTWARE FAULT-TOLERANCE EXERCISE: MECHANISMS I

Dmitrii Kuvaiskii
dmitrii.kuvaiskii@tu-dresden.de

Dresden, June 3, 2016

Outline

1. Warm-up
2. Mechanisms: Process Pairs, Graceful Degradation, Selective Retries
3. Live programming: robust C programming: echo server

Warm-up

Concepts

- What is robustness?
- What is a service?
- What interfaces have to be considered?
- What should a robust program handle?

Robustness and SFT

- Difference between SFT and robustness?
- What are environment failures?
- Why many services crash under high load?

Robustness Objectives

- What is masking?
- What is graceful degradation?
- What to do if one cannot mask nor graceful degradation?
- What should be specially avoided?

Bugs

- What are Bohrbugs?
- What are Heisenbugs?
- Sources of heisenbugs

Bugs

- What are Bohrbugs?
- What are Heisenbugs?
- Sources of heisenbugs
- Which is easier to handle and when?

Robustness mechanisms

Process Pairs

- What is the main purpose of process pairs?

Process Pairs

- What is the main purpose of process pairs?
- What is the programming construct to create process pairs?

TECHNISCHE UNIVERSITÄT DRESDEN

Process Pairs: Template

```
int ft = backup();
...
for(;;) {
  wait_for_request(Request);
  process_request(Request);
}
```

Process Pairs: The backup() function

```
int backup() {
  ...
  while(true) {
    ret = fork();
    ...
    if(ret == 0) {
      ...
      return ...;
    }
    waitpid(ret,0,0);
  }
}
```

Graceful Degradation

- What is graceful degradation?

Graceful Degradation

```
int backup() {
  ...
  while(true) {
    ret = fork();
    if(ret < 0) {
      return ...;
    } // parent goes into the loop
    if(ret == 0) {
      ...
      return ...;}
    waitpid(ret, 0, 0);
}}
```

Selective Retries

- What is the purpose of selective retries?
- Why does it work (eventually)?

Selective Retries

```
void* retry_malloc(size_t size) {
  void* buffer = NULL;
  int retries = 3, i = 0;
  struct timespec RETRY_SLEEP = {0, 50000000L};
  while (buffer == NULL) {
    buffer = malloc(size);
    if (NULL != buffer) break;          // no error
    nanosleep (&RETRY_SLEEP, NULL);     // wait a while
    i++;
    if ( i >= retries ) {
      perror(``failed to allocate memory'');
      exit(1);}}
  return buffer;}
```

Live programming

Example: echo server

- listens on TCP port 10000
- receives arbitrary data and sends it back to sender

Task for live programming:

1. incorporate process pair approach
2. automatically restart child process
3. set up pipe between parent and child
4. automatically restart parent process

TCP sockets

Common API for protocols like TCP, UDP, . . .

- Allocate a socket and associate it with a specific protocol
- Associate the socket with a local endpoint of communication
- Associate the socket and the local endpoint of communication with a remote endpoint of communication
- Transfer data
- Terminate local and remote endpoint associations, free the socket

Unix command line tools

- ps to list all processes
- kill to terminate a process
- grep to filter output
- killall to kill all processes with given name

Literature

- Beej's Guide to Network Programming (http://beej.us/guide/bgnet/)
- Beej's Guide to Unix Interprocess Communication (http://beej.us/guide/bgipc/)