Selected Debugging Topics

Debugging
- Debugging is the process of locating and removing errors.
  - It has been estimated that 85% of debugging time is spent locating errors and only 15% spent fixing them.
  - Tools and tips that speed up locating errors are very useful.

Common mistakes
- Typoes as `a = b` and `a == b`
- Uninitialised variables
- Case fall-through in a `switch` (w/o `break`)
- Pointer operations and memory leaks
- Array indexes are out of bounds
- Incorrect logic in conditions

Debugging at compile time
- Use `-Wall` switch
  
  ```
  $ gcc -Wall -o example example.c
  ```
  (in nachos Makefiles by default)

  - Analyze warnings (some are not serious)

Debugging with `printf()`
- Good places to add them in your code:
  - at the start of a function
    - to print input parameters
  - at the return point of a function
    - to print output parameters and the return value
  - at the start of a loop
    - print the loop counter and actions

`printf()`
- `printf()` must identify where it is in the code:
  - print the file name, the function name, id number, as well as data values
  - use adornments as [], _, <> etc.

- `printf()` may be conditional – use macros
Useful Preprocessor Macros:

- __DATE__
  - string "mmm dd yyyy"
- __FILE__
  - string with the current file name
- __LINE__
  - current line number
- __TIME__
  - string with the current time "hh:mm:ss"

Example: dprintf.c

```c
#include <stdio.h>

int main()
{
  int x = 5;
  printf("Compiled: %s %s\n", __DATE__, __TIME__);
  printf("%s/main.%d: x=%d\n", __FILE__, __LINE__, x);
  printf("hello world, x=%d\n", x);
  return 0;
}
```

Output

```
$ gcc -Wall -o dprintf dprintf.c
$ dprintf
Compiled: Dec 6 2000
10:27:44
dprintf.c/main.11: x=5
hello world, x=5
```

Conditional compilation

- You can switch off and on debugging printf’s commenting them around:

```c
\* Printf("Checkpoint 2 passed!\n" *
Too tedious!
```

- Possible to compile conditionally with or without debugging printouts – conditional compilation

Example: cinfo.c

```c
#include <stdio.h>

int main()
{
  int x = 5;
  #ifdef DEBUG
    printf("Compiled: %s %s\n", __DATE__, __TIME__);
    printf("%s/main.%d: x=%d\n", __FILE__, __LINE__, x);
  #endif
  printf("hello world, x=%d\n", x);
  return 0;
}
```

Output

```
$ gcc -Wall -o cinfo cinfo.c
$ cinfo
hello world, x=5
$ gcc -Wall -o cinfo cinfo.c -DDEBUG
$ cinfo
Compiled: Dec 6 2000 10:39:14
cinfo.c/main.15: x=5
hello world, x=5
```
DEBUG macro
- Defined in threads/utility.cc
  DEBUG(char flag, char* string, ...)
- Conditional print based on flag
- Run nachos with arguments –d flag or –d + for to enable all flags

ASSERT()
- For testing run-time assertions (conditions)
  ASSERT(filehandler = open(filename))
- If that expression evaluates to 0 at runtime then the program aborts with an error message
- Switch off with
  - #define NDEBUG (preprocessor directive)
  - -DNDEBUG (gcc argument)
- OS cannot just abort -> do something besides assertions

Debugging with libumem.
Theory
- In C/C++ some illegal memory operations are not caught during run-time (E.g. array bound violation or calling delete on the same pointer twice)
- It leads to memory corruption
- A program may crash in completely unrelevant place or finish with wrong results

Debugging with libumem
- libumem library helps to identify memory management bugs (there exists other software as well)
- Use especially if strange errors with new/delete operators are reported
- Run “setenv LD_PRELOAD libumem.so” and “ddd nachos” in the same terminal afterwards (works with tcsh and similar)
- DDD should abort now on the first run-time memory whenever discovered by libumem