CU CS 4995-03: Internet Systems Programming

Structuring System Software

Overview

- structuring code
- variable-length data
- common mistakes

Structuring Code

- library
- call-back
- threads
- message-based

Libraries

integration with different event models

- own set of threads
- select loop $\Longrightarrow$ can't add events:

```
tv.tv_sec = 10;
tv.tv_usec = 0;
while (1) {
  FD_ZERO(&rfds);
  FD_SET(s, &rfds);
  n = select(64, &rfds, NULL, NULL, &tv);
  if (n == 0) { printf("Timeout!\n"); }
  else if (n > 0) {
```

```
      if (FD_ISSET(s, &rfds)) {
        /* do something with thread */
      }
    }
  }
```

- add to `select` loop, e.g.:

  ```
  XtInputId XtAppAddInput(app_context, source, condition,
       proc, client_data)
  ```

- register callbacks, e.g., in Tcl: `socket -server` *command*
- C++ abstract classes with `virtual`, Java

Library Example: An Internet Phone API

- library for making (Internet) phone calls
- phone pickup may take a while: asynchronous vs. synchronous functions
- either side may terminate the call

Performance

No single performance measure for system program:

- first (startup; cgi!): response delay
- repeat: impact of $n + 1$ request
- streaming: data movement
- caching issues

Dealing with Variable-Length Data

"Write a C/C++ function that reads a string from a file (or returns some other variable-length value) and returns the string to the caller. The length of the string is unbounded and not known in advance."

- Who allocates memory?
- Who gets to write it?
- How long does it stay around?
- Java?

Approach 1: Static

```
const char *get_string(void)
{
  static char buf[10000];  /* be generous */
  char *s;

  return fgets(buf, sizeof(buf), stdin);
}
```

Common for C library: `asctime()`,

Problems:

-
-
-

Approach 2: Static Pointer to Dynamic Memory

```
const char *get_string()
{
  static char *result = 0;
  static size_t *rsize = 0;
  static const size_t size_of_block = 512;
  size_t rlen;

  rlen = 0;
  while (data left) {
    /* read a block of data */
    if (rsize - rlen < size_of_block) {
      rsize += size_of_block;
      result = realloc(result, rsize);
    }
```

```
    /* append block */
    rlen += size_of_block;
  }
}
```

Variation on a Theme: Dynamic Strings

```
typedef struct {
  char *string;
  int length;
  int spaceAvl;
} DString;

char *DStringAppend(DString *d, char *string, int length)
{
  if (length < 0) {
    if (string) length = strlen(string);
    else return d->string;
  }
  if (d->length + length + 1 >= d->spaceAvl) {
    if (!d->spaceAvl) {
      d->spaceAvl = length + 1;
```

```
      d->string = malloc(d->spaceAvl);
    }
    else {
      d->spaceAvl *= 2;
      if (d->spaceAvl < d->length + length + 1) {
        d->spaceAvl += length + 1;
      }
      d->string = realloc(d->string, d->spaceAvl);
    }
  }
  if (string) {
    memcpy(&d->string[d->length], string, length);
    d->length += length;
  }
  d->string[d->length] = '\0';
  return d->string;
} /* DStringAppend */
```

Approach 3: Caller-Allocated Memory

```
size_t get_string(char *result, size_t rsize)
{
  return read(0, result, rsize);
}
```

- common mechanism for all Unix system calls
-
-

Re-Entrant Functions

```
extern char *asctime_r(const struct tm *, char *, int)
extern char *ctime_r(const time_t *, char *, int);
extern char *tmpnam_r();
extern char *getlogin_r()
extern char *strtok_r();
```

Characteristics:

-
-

Approach 4: Return Pointer to Dynamic Memory

```
char *get_string()
{
  char *buf = 0;
  int size = 512;
  int n = 0;
  char *s;

  buf = malloc(size);
  while (1) {
    s = fgets(&buf[n], size, stdin);
    if (!s) break;
    if (buf[strlen(s)-1] == '\n') {
      break;
    }
```

```
    buf = realloc(buf, size += 512);
    n += 512;
  }
  return buf;
}
```

Approach 4: Return Pointer to Dynamic Memory

-
-
- `alloca()`
- `regexp(),...`

Common Mistakes

- ± 1 error
- access free'd memory
- write access to char constants (`strtok`)
-