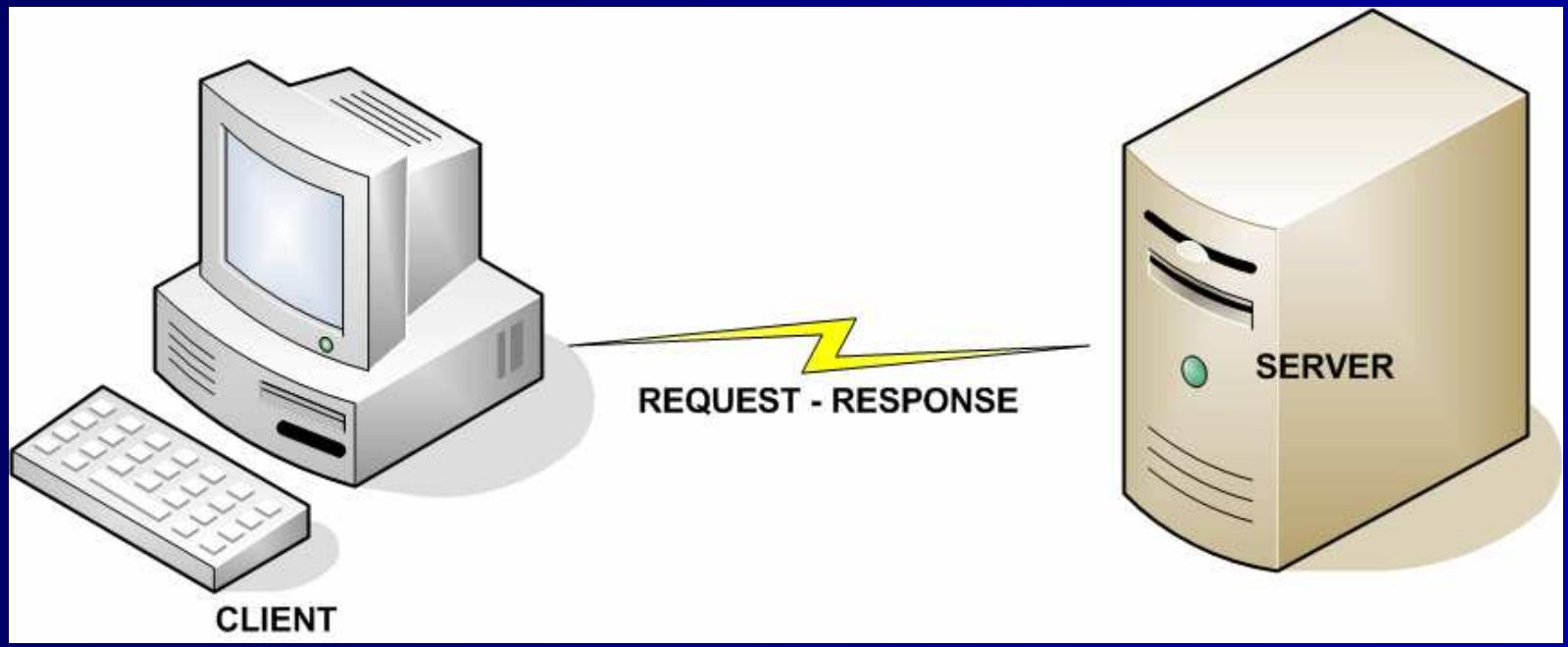
Ingineria Programării în Reţea (IPR)

Arhitectura Client-Server

Arhitectura Client-Server

- Foarte răspândită
- Potrivită pentru majoritatea aplicaţiilor de reţea
- Asigură *decuplarea temporală* a celor doi interlocutori (procese concurente)

Client-Server



Traffic pattern-uri

- 1 request – 1 response (interogare server)
- 1 request – multiple responses (cerere date + transfer de fişiere Server -> Client. Ex. HTTP)
- Multiple requests – 1 response (cerere upload date. Ex. FTP)

Stocare temporară

- Ce se întâmplă cu mai multe cereri care ajung în acelaşi timp la Server?
- Nevoia unei cozi la recepţie, pentru stocarea temporară a cererilor care aşteaptă să fie rezolvate
- Ce se întâmplă atunci când coada de recepţie se umple şi următoarele cereri se pierd?

Mai multe feluri de clienţi

- Cum facem distincţie între diversele tipuri de clienţi?
  - Folosind priorităţi
  - Şi un sistem de cozi + un algoritm de planificare

Procesare în paralel

- Dacă resursele serverului permit, atunci mai multe cereri pot fi procesate în paralel:
  - Servere multi-process / multi-tasking
  - Un proces (sau mai multe) dedicat primirii de cereri şi un proces (sau mai multe) dedicate procesării acestora

Performanţele unei aplicaţii Request-Response (Client-Server)

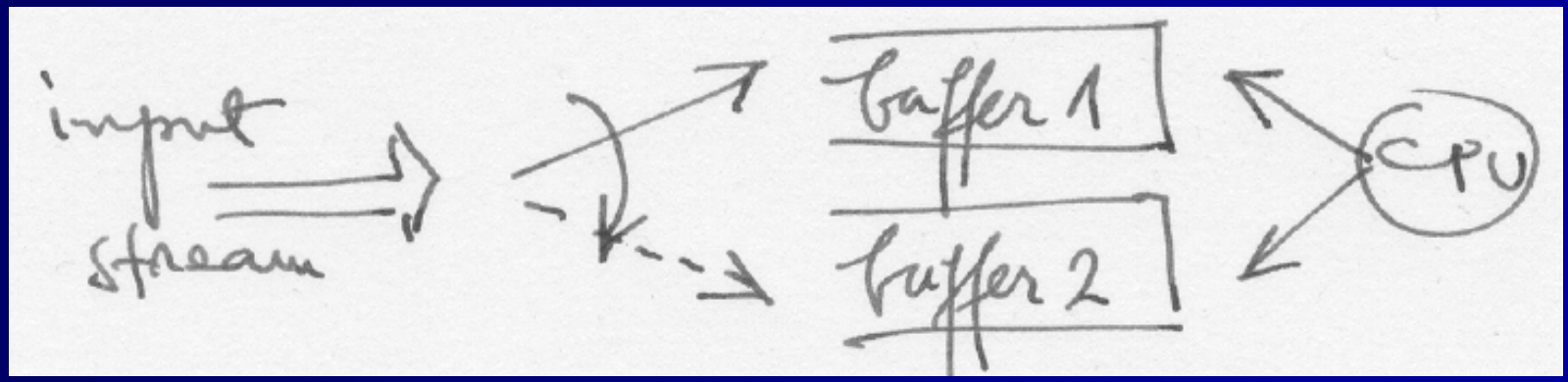
- Care este utilizarea conexiunii sau randamentul aplicaţiei cu o arhitectură Client-Server, în cazul în care se realizează mai multe transferuri (dinspre Client sau dinspre Server) şi se aşteaptă confirmarea primirii?
  - ţinând cont de RTT
  - sau în cazul în care timpul de procesare a cererii este de acelaşi ordin de mărime cu timpul de transfer a cererii de la Client la Server

Tehnici de îmbunătăţire a performanţei unei aplicaţii Client-Server

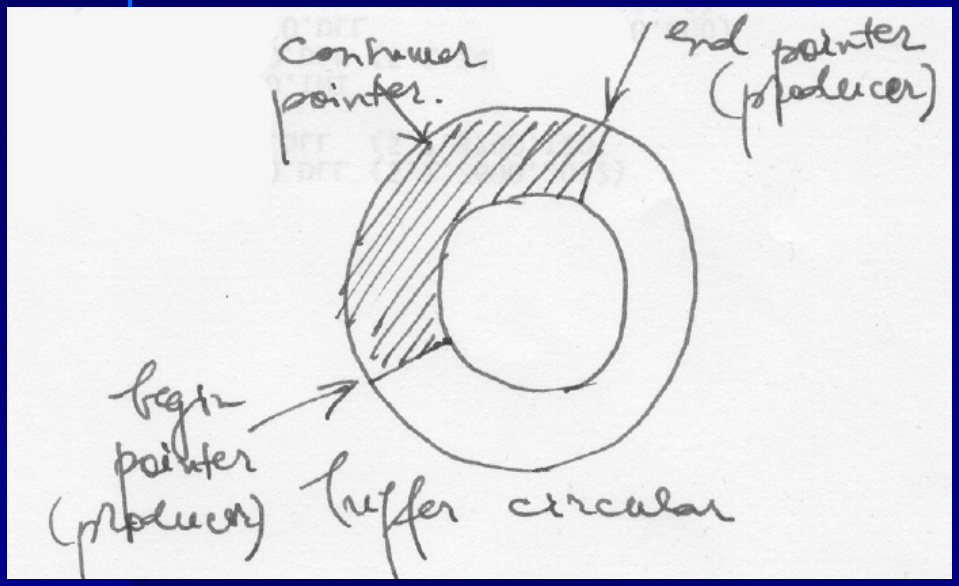
- Double buffering
  - Pentru mărirea debitului
- Circular buffer (ring buffer)
  - Pentru reducerea manipulării datelor din buffer (doar datele noi sunt scrise în buffer)

Double buffering

- În timp ce un buffer (o coadă) este umplut, celălalt este golit, debitul fiind astfel optimizat
- Similar cu tehnica "pipe-line"



Buffer circular



- Implementează o coadă infinită, cu observaţia că datele pot fi suprascrise
- Un mod de rezolvare a problemei Producer-Consumer
- Utilizat pentru implementarea TCP

Buffer circular - probleme

- Distincţia între cazurile “buffer plin” şi “buffer gol”, când ambii pointeri au aceeaşi valoare
  - Menţinerea unei locaţii goale tot timpul
  - Menţinerea evidenţei nivelului de umplere a bufferului (o variabilă în plus)
  - Utilizarea de indici cu valoare absolută (un bit în plus pentru pointeri)

Exemplu. Implementare POSIX

```
#include <sys/mman.h>
#include <stdlib.h>
#include <unistd.h>

#define report_exceptional_condition() abort ()

struct ring_buffer
{
   void *address;
   unsigned long count_bytes;
   unsigned long write_offset_bytes;
   unsigned long read_offset_bytes;
};
```

```
void ring_buffer_create (struct ring_buffer *buffer, unsigned long order)
{
    char path[] = "/dev/shm/ring-buffer-XXXXXX";
    int file_descriptor;
    void *address;  int status;

    file_descriptor = mkstemp (path);
    if (file_descriptor < 0) report_exceptional_condition ();

    status = unlink (path);
    if (status) report_exceptional_condition ();

    buffer->count_bytes = 1UL << order;
    buffer->write_offset_bytes = 0;
    buffer->read_offset_bytes = 0;
    status = ftruncate (file_descriptor, buffer->count_bytes);
    if (status) report_exceptional_condition ();

    buffer->address = mmap (NULL, buffer->count_bytes << 1, PROT_NONE,
    MAP_ANONYMOUS | MAP_PRIVATE, -1, 0);
    if (buffer->address == MAP_FAILED) report_exceptional_condition ();

    address = mmap (buffer->address, buffer->count_bytes, PROT_READ |
    PROT_WRITE, MAP_FIXED | MAP_SHARED, file_descriptor, 0);
    if (address != buffer->address) report_exceptional_condition ();
    address = mmap (buffer->address + buffer->count_bytes, buffer->count_bytes,
    PROT_READ | PROT_WRITE, MAP_FIXED | MAP_SHARED, file_descriptor, 0);
    if (address != buffer->address + buffer->count_bytes)
    report_exceptional_condition ();
    status = close (file_descriptor);
    if (status) report_exceptional_condition ();
}
```

```
void ring_buffer_free (struct ring_buffer *buffer)
{
   int status;
   status = munmap (buffer->address, buffer->count_bytes <<
   1);

   if (status)
        report_exceptional_condition ();
}

void * ring_buffer_write_address (struct ring_buffer
   *buffer)
{
   return buffer->address + buffer->write_offset_bytes;
}

void ring_buffer_write_advance (struct ring_buffer *buffer,
   unsigned long count_bytes)
{
   buffer->write_offset_bytes += count_bytes;
}
```

```
void * ring_buffer_read_address (struct ring_buffer
   *buffer)
{
   return buffer->address + buffer->read_offset_bytes;
}
```

```
void ring_buffer_read_advance (struct ring_buffer
   *buffer, unsigned long count_bytes)
{
   buffer->read_offset_bytes += count_bytes;

   if (buffer->read_offset_bytes >= buffer-
   >count_bytes)
   {
     buffer->read_offset_bytes -= buffer->count_bytes;
      buffer->write_offset_bytes -= buffer-
   >count_bytes;
   }
}
```

```
unsigned long ring_buffer_count_bytes (struct
   ring_buffer *buffer)
{
   return buffer->write_offset_bytes - buffer-
   >read_offset_bytes;
}

unsigned long ring_buffer_count_free_bytes (struct
   ring_buffer *buffer)
{
   return buffer->count_bytes -
   ring_buffer_count_bytes (buffer);
}

void ring_buffer_clear (struct ring_buffer *buffer)
{
   buffer->write_offset_bytes = 0;
   buffer->read_offset_bytes = 0;
}
```

Buffer overrun

- Fenomen de suprascriere ce poate apărea la implementarea unei cozi
- Punctul sensibil: copierea de şiruri de caractere, într-un mod *neglijent*
- Un mod de atac al aplicaţiei des utilizat de "hackeri" şi o modalitate de implementare a viruşilor: şir de caractere reprezentând codul executabil al unui virus, ajunge să se suprascrie peste zona de memorie a unei funcţii

Buffer overrun

```
void myMethod( char * pStr )
{
    char pBuff[ 10 ];
    int nCount = 0;

    strcpy( pBuff, pStr );
}

void foo()
{
}
```