Unit 4: Memory Management

4.5. Win32 Inter-process Communication

Win32 Interprocess Communication

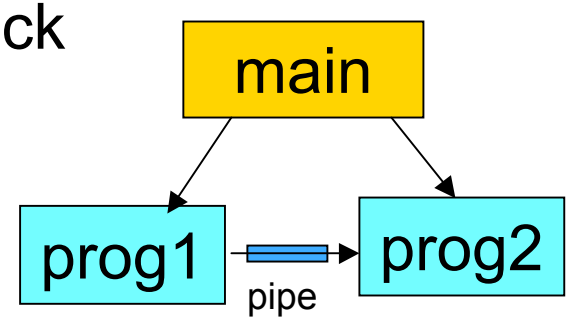
- Anonymous pipes
  - Character-based, half-duplex
  - Input/output redirection (cat file | more)
- Named pipes
  - Full-duplex, message-oriented
  - Allow networked communication (!)
  - Transaction-oriented named pipe functions
- Mailslots
  - One-to-many message broadcasting

Anonymous pipes

```
BOOL CreatePipe( PHANDLE phRead,
                 PHANDLE phWrite,
                 LPSECURITY_ATTRIBUTES lpsa,
                 DWORD cbPipe )
```

Half-duplex character-based IPC

- cbPipe: pipe byte size; zero == default
- Read on pipe handle will block if pipe is empty
- Write operation to a full pipe will block
- Anonymous pipes are oneway



I/O Redirection using an Anonymous Pipe

```
/* Create an anonymous pipe with default size. The handles are inheritable. */
if (!CreatePipe (&hReadPipe, &hWritePipe, &PipeSA, 0))
        ReportError (_T ("Anon pipe create failed."), 3, TRUE);

/* Set the output handle to the inheritable pipe handle,
        and create the first processes. */
StartInfoCh1.hStdInput  = GetStdHandle (STD_INPUT_HANDLE);
StartInfoCh1.hStdError  = GetStdHandle (STD_ERROR_HANDLE);
StartInfoCh1.hStdOutput = hWritePipe;
StartInfoCh1.dwFlags = STARTF_USESTDHANDLES;

if (!CreateProcess (NULL, (LPTSTR)Command1, NULL, NULL, TRUE, /* Inherit
handles. */
                        0, NULL, NULL, &StartInfoCh1, &ProcInfo1)) {
        ReportError (_T ("CreateProc1 failed."), 4, TRUE);
}
CloseHandle (hWritePipe);
```

Pipe example (contd.)

```
/* Repeat (symmetrically) for the second process. */
StartInfoCh2.hStdInput  = hReadPipe;
StartInfoCh2.hStdError  = GetStdHandle (STD_ERROR_HANDLE);
StartInfoCh2.hStdOutput = GetStdHandle (STD_OUTPUT_HANDLE);
StartInfoCh2.dwFlags = STARTF_USESTDHANDLES;

if (!CreateProcess (NULL, (LPTSTR)targv, NULL, NULL,TRUE,/* Inherit handles. */
                    0, NULL, NULL, &StartInfoCh2, &ProcInfo2))
        ReportError (_T ("CreateProc2 failed."), 5, TRUE);
CloseHandle (hReadPipe);

/* Wait for both processes to complete. */
WaitForSingleObject (ProcInfo1.hProcess, INFINITE);
WaitForSingleObject (ProcInfo2.hProcess, INFINITE);
CloseHandle (ProcInfo1.hThread); CloseHandle (ProcInfo1.hProcess);
CloseHandle (ProcInfo2.hThread); CloseHandle (ProcInfo2.hProcess);
return 0;
```

Named Pipes

- Message oriented:
  - Reading process can read varying-length messages precisely as sent by the writing process
- Bi-directional
  - Two processes can exchange messages over the same pipe
- Multiple, independent instances of a named pipe:
  - Several clients can communicate with a single server using the same instance
  - Server can respond to client using the same instance
- Pipe can be accessed over the network
  - location transparency
- Convenience and connection functions

Using Named Pipes

```
HANDLE CreateNamedPipe (LPCTSTR lpszPipeName,
        DWORD fdwOpenMode, DWORD fdwPipMode
        DWORD nMaxInstances, DWORD cbOutBuf,
        DWORD cbInBuf, DWORD dwTimeOut,
        LPSECURITY_ATTRIBUTES lpsa );
```

- lpszPipeName: \\.\pipe\[path]pipename
  - Not possible to create a pipe on remote machine (. – local machine)
- fdwOpenMode:
  - PIPE_ACCESS_DUPLEX, PIPE_ACCESS_INBOUND, PIPE_ACCESS_OUTBOUND
- fdwPipeMode:
  - PIPE_TYPE_BYTE or PIPE_TYPE_MESSAGE
  - PIPE_READMODE_BYTE or PIPE_READMODE_MESSAGE
  - PIPE_WAIT or PIPE_NOWAIT (will ReadFile block?)

Use same flag settings for all instances of a named pipe

Named Pipes (contd.)

- nMaxInstances:
  - Number of instances,
  - PIPE_UNLIMITED_INSTANCES: OS choice based on resources
- dwTimeOut
  - Default time-out period (in msec) for WaitNamedPipe()
- First CreateNamedPipe creates named pipe
  - Closing handle to last instance deletes named pipe

- Polling a pipe:
  - Nondestructive – is there a message waiting for ReadFile

```
BOOL PeekNamedPipe (HANDLE hPipe,
        LPVOID lpvBuffer, DWORD cbBuffer,
        LPDWORD lpcbRead, LPDWORD lpcbAvail,
        LPDWORD lpcbMessage);
```

Named Pipe Client Connections

- CreateFile with named pipe name:
  - \\.\pipe\[path]pipename
  - \\servername\pipe\[path]pipename
  - First method gives better performance (local server)
- Status Functions:
  - GetNamedPipeHandleState
  - SetNamedPipeHandleState
  - GetNamedPipeInfo

Convenience Functions

- WriteFile / ReadFile sequence:

```
BOOL TransactNamedPipe( HANDLE hNamedPipe,
        LPVOID lpvWriteBuf, DWORD cbWriteBuf,
        LPVOID lpvReadBuf, DWORD cbReadBuf,
        LPDOWRD lpcbRead, LPOVERLAPPED lpa);
```

- CreateFile / WriteFile / ReadFile / CloseHandle:
  - dwTimeOut: NMPWAIT_NOWAIT, NMPWAIT_WIAT_FOREVER, NMPWAIT_USE_DEFAULT_WAIT

```
BOOL CallNamedPipe( LPCTSTR lpszPipeName,
        LPVOID lpvWriteBuf, DWORD cbWriteBuf,
        LPVOID lpvReadBuf, DWORD cbReadBuf,
        LPDWORD lpcbRead, DWORD dwTimeOut);
```

Server: eliminate the polling loop

BOOL ConnectNamedPipe (HANDLE hNamedPipe,
LPOVERLAPPED lpo

NT only

- Lpo == NULL:
  - Call will return as soon as there is a client connection
  - Returns false if client connected between CreateNamed Pipe call and ConnectNamedPipe()
- Use DisconnectNamedPipe to free the handle for connection from another client
- WaitNamedPipe():
  - Client may wait for server‘s ConnectNamedPipe()
- Security rights for named pipes:
  - GENERIC_READ, GENERIC_WRITE, SYNCHRONIZE

Comparison with UNIX

- UNIX FIFOs are similar to a named pipe
  - FIFOs are half-duplex
  - FIFOs are limited to a single machine
  - FIFOs are still byte-oriented, so its easiest to use fixed-size records in client/server applications
  - Individual read/writes are atomic
- A server using FIFOs must use a separate FIFO for each client‘s response, although all clients can send requests via a single, well known FIFO
- Mkfifo() is the UNIX counterpart to CreateNamedPipe()
- Use sockets for networked client/server scenarios

Client Example using Named Pipe

```
WaitNamedPipe (ServerPipeName, NMPWAIT_WAIT_FOREVER);
hNamedPipe = CreateFile (ServerPipeName, GENERIC_READ | GENERIC_WRITE,
        0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
if (hNamedPipe == INVALID_HANDLE_VALUE)
        ReportError (_T ("Failure to locate server."), 3, TRUE);

        /* Write the request. */
WriteFile (hNamedPipe, &Request, MAX_RQRS_LEN, &nWrite, NULL);

        /* Read each response and send it to std out. */
while (ReadFile (hNamedPipe, Response.Record, MAX_RQRS_LEN, &nRead, NULL))
        _tprintf (_T ("%s"), Response.Record);

CloseHandle (hNamedPipe);
return 0;
```

Server Example Using a Named Pipe

```
hNamedPipe = CreateNamedPipe (SERVER_PIPE, PIPE_ACCESS_DUPLEX,
            PIPE_READMODE_MESSAGE | PIPE_TYPE_MESSAGE | PIPE_WAIT,
            1, 0, 0, CS_TIMEOUT, pNPSA);
while (!Done) {                     /* Main Command Loop. */
            _tprintf (_T ("Server is awaiting next request.\n"));
            if (!ConnectNamedPipe (hNamedPipe, NULL)
                        || !ReadFile (hNamedPipe, &Request, RQ_SIZE, &nXfer, NULL))
                        ReportError (_T ("Connect or Read Named Pipe error."), 0, TRUE);
                        _tprintf (_T ("Request is: %s\n"), Request.Record);
            /* Send the temp file, one line at a time, to the client. */
            fp = _tfopen (TempFile, _T ("r"));
            while ((_fgetts (Response.Record, MAX_RQRS_LEN, fp) != NULL))
                        WriteFile (hNamedPipe, &Response.Record,
                                    (_tcslen (Response.Record) + 1) * TSIZE,&nXfer, NULL);
            fclose (fp);
            DisconnectNamedPipe (hNamedPipe);
}    /* End of while loop and server operation. */
```