I3: 情報（第 3 部）

03-160471 井上友貴 / 03-160441 土屋潤一郎（第 28 班 A）

---

1. 実装の概要

- UDP 通信による音声電話の実現
  – 無音判定による通信量の抑制
- OpenAL による録音・再生
- ファイル送受信
- POSIX スレッドによるこれらのマルチスレッド化
- popen 関数と system 関数による一部マルチプロセス化

2. 実装の詳細

2.1 “電話”

電話とは何か，すなわち，人間側から見てどのような手順で通信がつなげられ，どのような情報が，どのようにやりとりされるのか．そのために最低限必要なのはどのような機能なのか，を考え，それを再優先で実装した．

2.1.1 電話を掛ける / 電話を取る

電話を掛けるとき，人は，受話器を取り，相手の電話番号をダイヤルして（有れば，電話帳機能を用いるかもしれない），呼び出し音を鳴らす．そして相手が出るのを待つ．基本的にはこの動作を再現したい．

IP 電話に於いては，電話番号は IP アドレスで，受話器はハンズフリーだから無いが，ソケットといえるかもしれない．

電話が電話線に繋がれているという前提の下，着信音が鳴る．これが”電話を取る”操作の開始である．
人は，着信音を鳴らす電話の受話器を取る．これが電話に出ることを了承した瞬間である．

IP 電話なので，電話線ではなくサーバー側ソケットを立てておくことになるだろう．PC なので着信音ではなく画面による通知でも良いであろう．ハンズフリーなので受話器は取らないが，電話を掛けられる側の人間が「電話に出ることを決意する瞬間」（「電話に出ることを拒否できる瞬間」とも言い換えられる）が必要だろう．出来れば相手の情報を得て（即ち番号通知機能である），電話に出るかどうか決定したい．

2.1.2 通話開始までの実装

前提として，通信には予め決まった番号のポートを用いる．

まず，プログラムの最初にサーバー側ソケットを立ち上げておく．これで，いつでも電話を掛けられることが出来る．

電話を掛ける側から始めることにする．
ユーザーが電話を掛けることを選択すると，まずサーバー側として立てていたソケットを閉じて，ポートを解放する．そして，（同じポートで）クライアントを立ち上げる．次いで，電話を掛ける相手の情報（ここでは IP アドレス）を標準入力を通じて受け取る．この IP アドレスに向けて，1 を 1000 バイト送信する．

電話を掛けられる側では，（自分から掛けるという動作を行っていなければ）常に UDP サーバー側ソケットがパケットの到着を待っている．パケットが到着すると，1000 バイトを走査し，中身が全て 1 ならその時点で標準出力に通知を出す．このとき，UDP ヘッダに含まれる相手の IP アドレス情報を，通知とともに標準出力に乗せる．

電話を掛けられた側のユーザは，通知を見て，電話に出るかどうかを決定出来る．電話に出ると決定し，それが標準入力を通じてプログラムに伝わると，サーバーが 1 を 1000 バイト送信し返す．

クライアントは，相手からの 1000 バイトの 1 によって，相手が電話に出たこと知る．ユーザはこれを標準出力への通知で知る．

これで，ユーザ二人が「相手が電話の前にいる」ということを互いに確認できた．従って通話の開始である．

2.1.3 通話

通話とはなんであろうか．聴覚のみを頼りにした会話である，と考えた．
会話で伝えられるべき情報はなんであろうか．
一方で，会話で欠損しても良い（あるいは，その欠損をすぐに補うことができる）情報とはなんであろうか．

会話で伝えられるべき情報は，つまり発話された内容である．それに加えて，声色もある程度伝えられるべきだろう．これらが，即時的に相手に伝わることが必要である．

一方で，欠損しても良い情報な何であろうか．雑音は欠損しても良い（というかむしろ，積極的に欠損させたい）．それから，内容に差し障りの無い範囲で，話者の発した声のデータも削りたい．

そして，即時性さえ確保されれば，多少の内容の欠損は，大概の場合補えるだろう．聞こえなかった内容は，即座に聞き返すことが可能である．

そもそも，単に PCM データを送ろうとする場合，まずは一定の標本数を録音せねばならない．この録音時間はダイレクトに遅延になるから，ある一定の時間で切り上げたい．そしてこの標本数を送信するとき，単位時間辺りの標本数が少ないほうが，送信するデータ量が少なくてすむ．

2.1.4 通話に関わる機能

本項では，録音されたデータと受信したデータをどのように扱うのかを説明する．
録音と再生については後述する．

正確には，録音と本項で説明する内容については分離しがたい関係がある．すなわち，どれだけの標本数の PCM データを 1 セットとして考えるかということについては，録音を司る OpenAL の API によって行われている．しかしここでは，とにかく一定の標本数が揃うと，それが PCM データとして配列に格納されるということがスタートである．

録音されたデータが送信されるまでに挟まれるのは，無音判定の段階である．一定以上の振幅を一つも持たないデータセットは，そもそも送らないことで，通信路に対する負荷を軽減する．さらにこれを以って，ハンズフリーで特に大きな問題となる音響エコーへの対策とする．

音響エコーが無音判定で軽減されうるのは，以下の条件が満たされる時である．

- 片方のユーザの声がもう一方のスピーカーから再生されたとき，そのユーザが発声していない．
- ユーザが常にエコーよりも大きな声で話す．

これらが満たされれば，エコーと思わしき振幅までを無音判定することで，比較的簡単にエコーをカットすることが出来る．
そしてこれらの条件は蓋然性が高い．すなわち，

- コミュニケーションに於いては相手の発言を最後まで聞いて，それから自らの発言を為すことが多いであろう．
- 大きな声で話せない環境であるならば，イヤホンとマイクを用いているであろう．

両者が同時に話すことを考えてエコー除去を行う適応フィルタは，残念ながらこのペアの実装力では能わなかった．また，エコー以外の（数学的な意味ではない）定常的な雑音については，これを除去する必要を感じなかった．スペクトルサブトラクション法は適応フィルタに比べれば簡易で，実装可能範囲にあるように思われたが，優先順位が低かった．（優先順位については，発表とスライドに譲る．）

一方，受信されたデータは，実のところ，何も加工されずにただストリーミング再生のシステムに流されるだけである．

2.1.5 録音と再生

録音と再生は，OpenAL による．そもそも OpenAL は，

> OpenAL is **a cross-platform 3D audio API** appropriate for use with gaming applications and many other types of audio applications.

で，

> The library models a collection of audio sources moving in a 3D space that are heard by a single listener somewhere in that space.

ということ（https://www.openal.org/ より）なので，録音に関しては（再生と比較すれば）簡素な機能があるのみである．

録音に関しては，開始したら，

- デバイスのバッファに対して，有効な標本数を尋ね続ける．
- 有効な標本数が一定の数以上になれば，
  – それを取り出してきて配列に格納する．

ということをひたすら繰り返すだけである．

再生に関してはまず，OpenAL の再生に関する基本的な枠組みを説明する．
OpenAL の基本的なオブジェクトとして，コンテキストの中に一人のリスナがいて，一つ以上のソースがある．ソースには一つ以上のバッファが割り当てられ，バッファは音源データを持っている．
実装者の理解した方法で喩えれば，コンテキストは，リスナ，つまり聴衆が一人しかいないホールである．そこには楽器，あるいはその奏者にあたるソースがあり，ソースはそれぞれ楽譜を読んで楽譜通りに音を出す．その楽譜（の１ページ）がバッファであって，その楽譜に記されている音符が，一つ一つの PCM データである．

これらによって便利に実現される立体音響に関しては，少なくとも 1on1 の電話ではあまり必要がない．今回の実装では，リスナとソースの位置関係に関してはデフォルトでよしなにやってもらうことにした．

音源割当の問題に移る．もっとも単純な音源再生は，PCM データの配列をバッファに割り当て，そのバッファをソースに再生させるという手順を踏むことになる．しかし今回は継続的にやってくるデータをストリーミング再生したい．
このために用意されているのがバッファのキューイングである．ソースはキューを持ち，ここにエンキューされた順に楽譜（バッファの配列）のページ（バッファ）を読んでいく．そして読み終わって鳴らし終わったページはデキュー出来る．

2.2 ファイル送受信

本実験では電話をしている最中にもなにがしかのことが出来ることを目指した．充分大きいバイト数を持つ情報は，音声でやり取り出来ない．従って，電話中に「該当のファイル送るよ〜」などと言ってファイルのやり取りをする場面があると便利そうなのは容易に想像がつく．

本実験ではこれを system 関数による nc コマンドの呼び出しで解決した．
送信側ユーザが送りたいファイル名と宛先を標準入力を通じて与え，受信側ユーザはそれに先んじてファイルの保存名を与えておく．受信側では nc コマンドのサーバが立ち（受信側が先んじていなければならない所以である），送信側が nc コマンドでクライアントとなってファイルを送信する．

2.3 チャット

チャットを（やはり system 関数を利用して）ターミナルの別のウィンドウで開き，TCP 通信のソケットプログラミングを用いてチャットを実装する予定であったが，TCP ソケットに関する性質を把握できていなかったため，叶わなかった．すなわち，UDP での電話とほとんど同様の構造でデータをやり取りしようとしたが，TCP ではソケットが送信と受信を交互にしか行えないという点がすっかり記憶から欠落していた．
従ってこの機能は未実装のまま，発表日を迎えることとなった．

2.4 POSIX スレッドによるマルチスレッド化とその他マルチプロセス化

これまで述べたとおり，ファイルの送受信やチャットはマルチプロセスによって並列処理が行われている．一方，電話本体（送受信や録音・再生）や，電話中にそれらの system 関数を呼び出せるようにする部分は，一部の例外（再生）を除いてマルチスレッド化している．

録音と送信が一つのスレッド，受信が一つのスレッドで，再生が popen 関数で別プロセスとして立ち上がる．またこれらの他に，標準出力にメニューを表示し標準入力で操作を受け付けるスレッドがある．

3. 考察

3.1 作業過程について

終わってから振り返れば，スライドでも紹介している通り，優先順位を最初に確定させたのは完全に正しかったといえる．実装力の貧弱なペアとしては，まず最低限の 1on1 の音声電話を完成させることが，実はかなりの重荷であった．

一方で反省すべき点は，2 つある．

第一に，OpenAL という，実装者にとって未知のライブラリを採用したことである．この採用は，多者間通話への拡張性・応用性を考慮して決定したが，これは誤りであったといえる．今後，短期間で成果を求められる際には，実装力の無い者は特に，学習する期間が無いものと心得ておくこととする．

第二に，最初に並列で行いたい処理全てをマルチスレッド化しようと決心したことである．このためにソースコードは長大となり，デバッグ作業は混迷を極め，作業時間は無為に失われていった．挙句の果てに，再生部分は単体では正常に動作するがスレッドとして組み込むとなぜかエラーを吐き，ついに別プロセスとして分割することとなった．発表では別の班が大量の分割されたプログラムをパイプで繋ぎあわせており，やはりそれが正しかったのだと強く感じた．

4. 参考書籍

1. 東京大学工学部電子情報工学科・電気電子工学科（2016）『電気電子情報第一（前期）実験 テキスト』
2. OpenAL Programmer's Guide

https://www.openal.org/documentation/OpenAL_Programmers_Guide.pdf