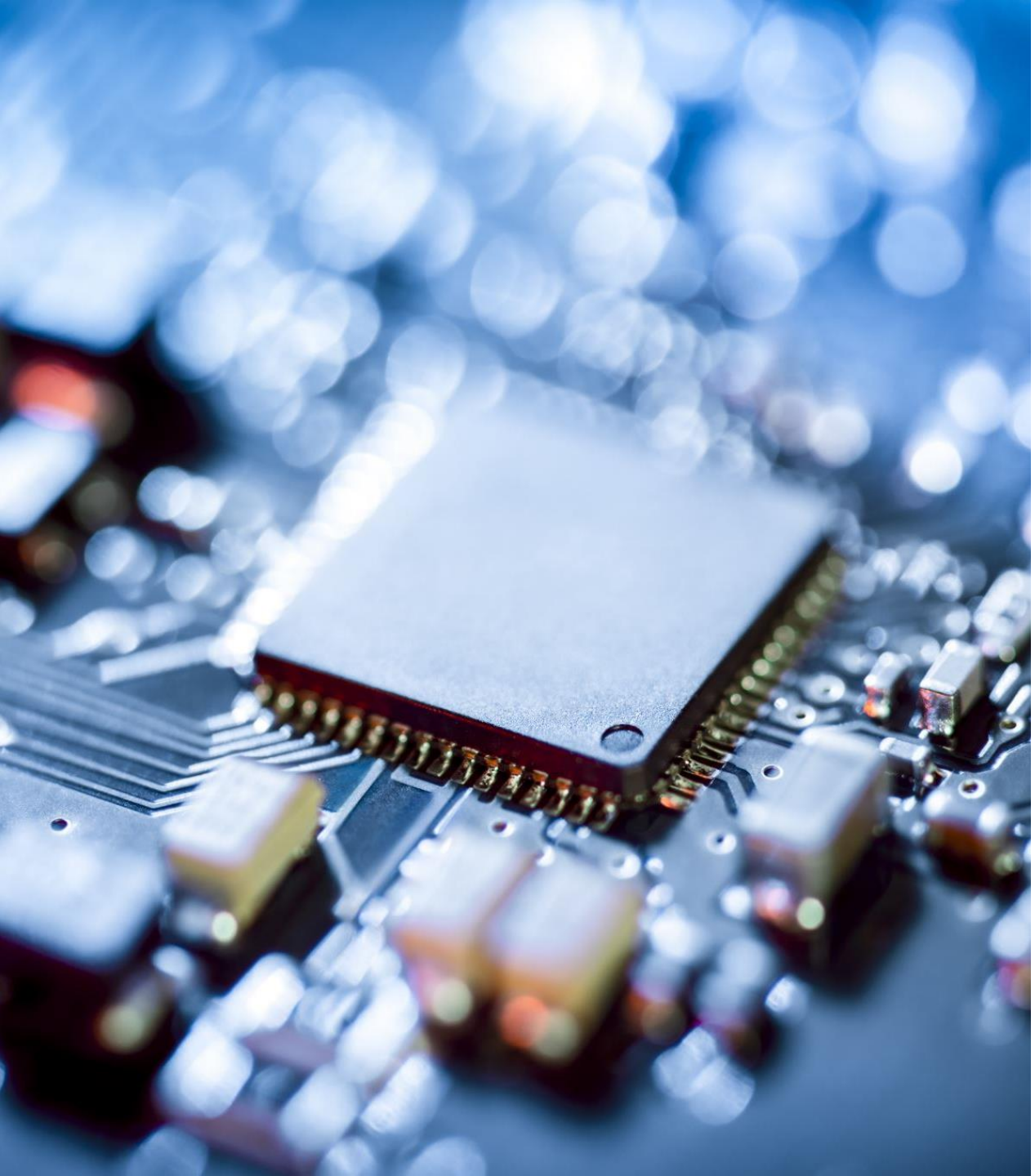


Sistemas Multiprocessados em Chip – Parte1

(03/06/24)

Alunos:

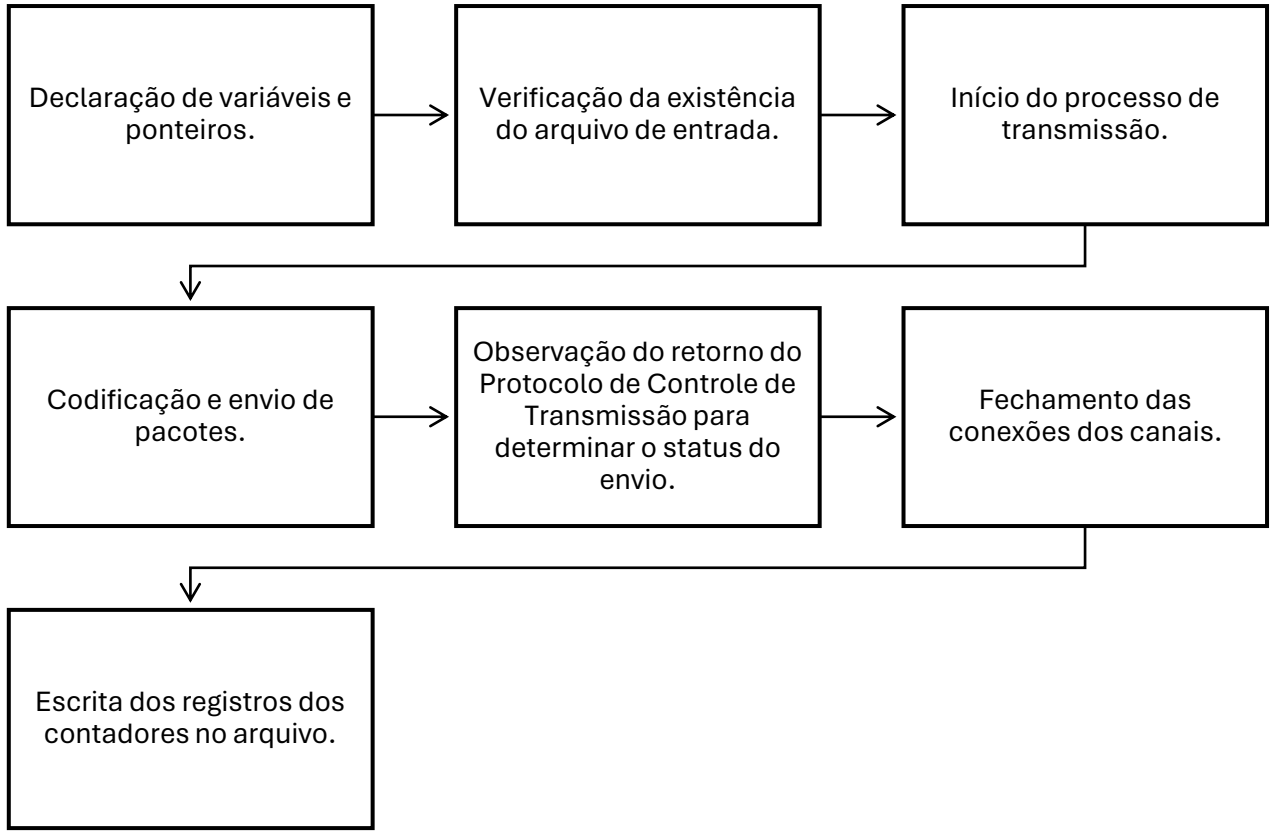
- Manuel Adahil Muniz Osterno
- Marco Vinícius Matos Oliveira

Professores:

- Dr. César Augusto Missio Marcon
- Dr. Fernando Gehm Moraes

Processador Alfa (1_proc_alfa.c)

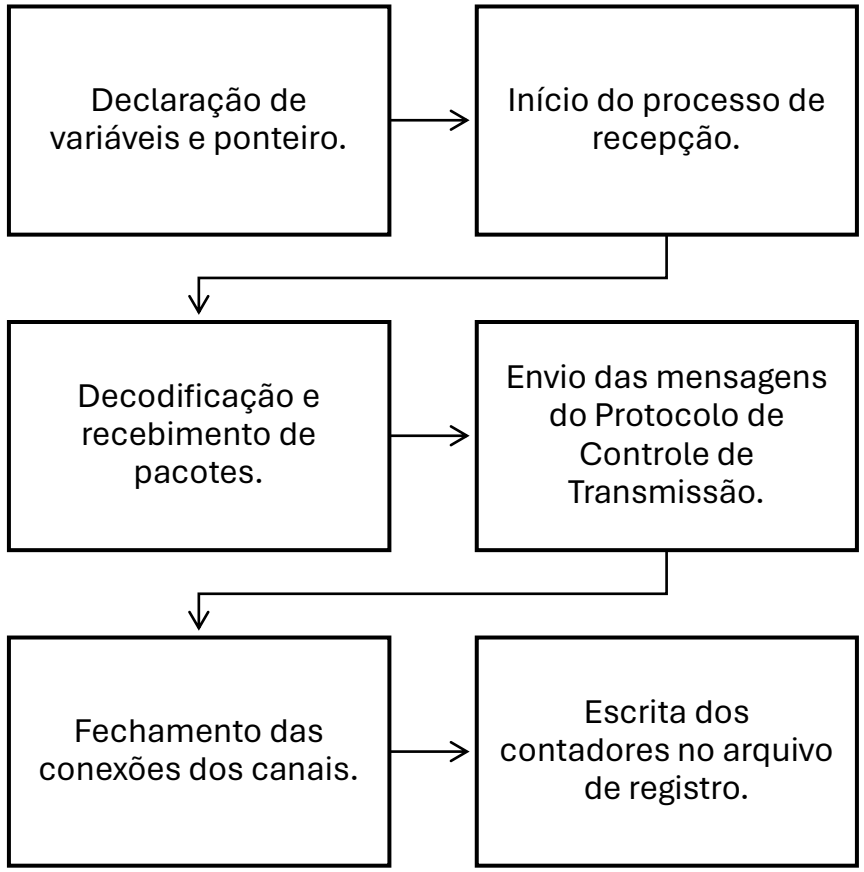
Transmissão de pacotes e Recebimento de respostas do Protocolo de Controle de Transmissão.




```c
#include <stdlib.h>
#include <stdio.h>
#include "packet.h"

int main (){
   uint32_t payload [PAYLOAD_SIZE];
   uint32_t pkt_nb = 0;
   packet_t packet;
   ackno_packet_t reply;
   connection_t con_inject;
   connection_t con_beta;
   uint8_t file_content[PAYLOAD_SIZE*4];
   FILE * input_fptr;
   FILE * log_fptr;

   //Statistics
   uint32_t nb_ack    = 0;
   uint32_t nb_ack_err = 0;
   uint32_t nb_nack   = 0;

   input_fptr = fopen("input_file", "r");

   if(input_fptr == NULL) { //Check if file exists
    printf("[PROC_ALFA] Not able to open the file.");
    return -1;
   }

   printf("[PROC_ALFA] Starting Process Alfa ...\n");
   printf("[PROC_ALFA] Waiting connection with Error Injector...\n");
   connect(&con_inject, CHANNEL_0, "ab+"); //Channel 0 is the communication channel between Alfa and
Error Injector
   printf("[PROC_ALFA] Waiting connection with Beta...\n");
   connect(&con_beta, CHANNEL_2, "rb+"); //Channel 2 is the communication channel between Alfa and Beta
   //@NOTE: I developed a version where the process run forever and wait to get something from file.
   //However, it needs to have a control to close and reopen the file several time, as this is not the
   //goal of the project, I gave up on this and did this version that runs until end of file.
   while(fgets(file_content, PAYLOAD_SIZE*4, input_fptr)) {
      printf("[PROC_ALFA] Waiting for reading file ...\n");
      encode_packet((uint32_t * ) file_content, &packet, PAYLOAD_SIZE, 2024, pkt_nb);
      print_packet(packet, PAYLOAD_SIZE);
      send_pckt(packet, &con_inject);
      printf("[PROC_ALFA] Waiting for acknowledgement ...\n");
      recv_ackno_reply(&reply, &con_beta);
      while (reply == NACK){
       nb_nack++;
       pkt_nb = pkt_nb;
       send_pckt(packet, &con_inject);
       recv_ackno_reply(&reply, &con_beta);
      }
      if (reply == ACK_ERR){
       nb_ack_err++;
       pkt_nb ++;
      } else {
       nb_ack++;
       pkt_nb ++;
      }
   }
   printf("Statisticas:\n");
   printf("N# ACK     : %d\n", nb_ack);
   printf("N# ACK ERROR: %d\n", nb_ack_err);
   printf("N# NACK    : %d\n", nb_nack);
   printf("\n");
   close_connect(&con_inject);
   close_connect(&con_beta);
   fclose(input_fptr);

   //Writing Statistics
   log_fptr = fopen("alfa_log_file", "w+");
   fprintf(log_fptr, "Statisticas:\n");
   fprintf(log_fptr,"N# ACK     : %d\n", nb_ack);
   fprintf(log_fptr,"N# ACK ERROR: %d\n", nb_ack_err);
   fprintf(log_fptr,"N# NACK    : %d\n", nb_nack);
   fclose(log_fptr);

   printf("[PROC_ALFA] Finishing Process Alfa ...\n");
   return 0;
}
```

Processador Beta (2_proc_beta.c)

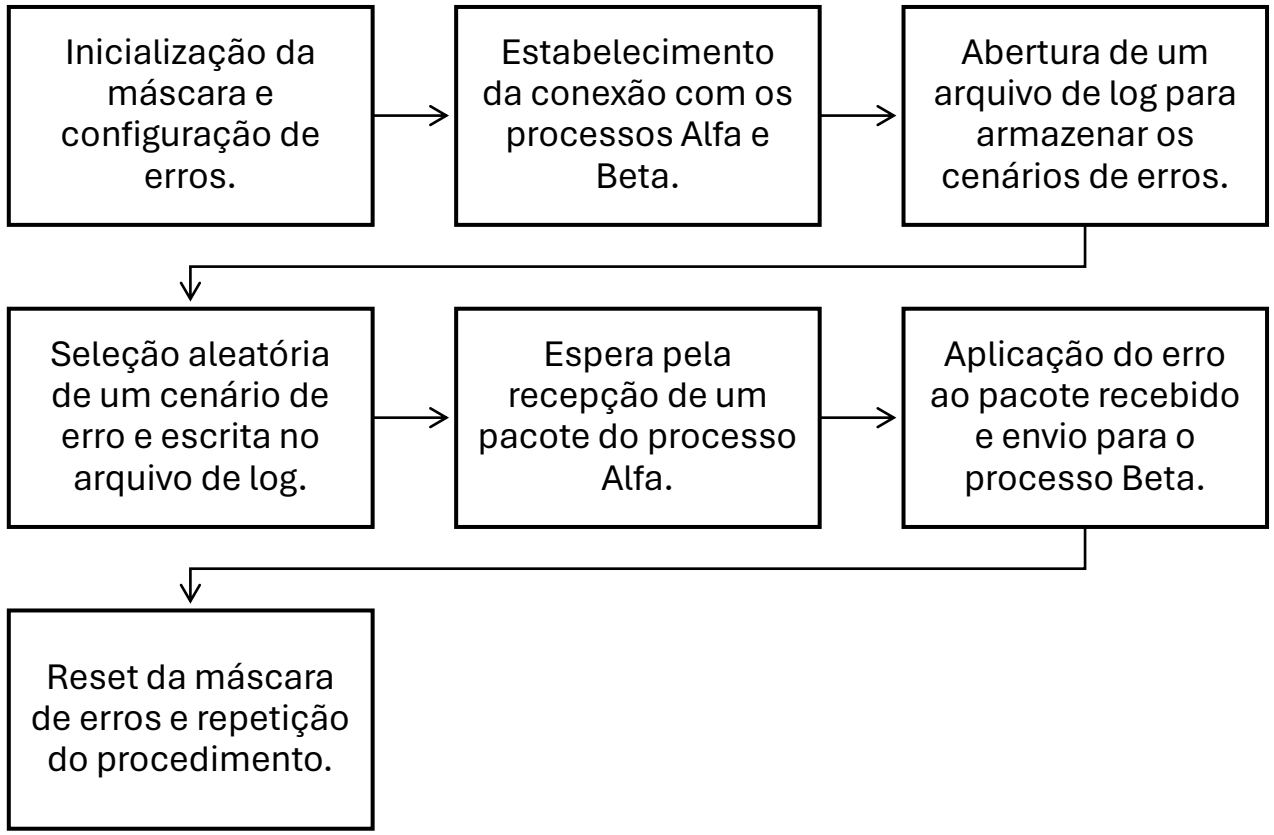
Recebimento de pacotes e Envio de respostas do Protocolo de Controle de Transmissão.




```
#include "packet.h"

int main(){
   uint32_t payload [PAYLOAD_SIZE];
   packet_t packet;
   pkt_error_t packet_err;
   uint32_t pkt_nb = 0;
   connection_t con_inject;
   connection_t con_alfa;
   FILE * output_fptr;

   printf("[PROC_BETA] Starting Process Beta ...\n");
   printf("[PROC_BETA] Waiting connection with Alfa...\n");
   connect(&con_alfa, CHANNEL_2, "ab+");//Channel 2 is the
communication channel for acknowledgement between Beta and Alfa
   printf("[PROC_BETA] Waiting connection with Error Injector...\n");
   connect(&con_inject, CHANNEL_1, "rb+");//Channel 1 is the
communication channel between Beta and Error Injector
   while(1) {
      output_fptr = fopen("output_file", "a+"); //Open output file
      printf("[PROC_BETA] Waiting for packet ...\n");
      recv_pckt(&packet, &con_inject);
      printf("[PROC_BETA] Packet before correction:\n");
      print_packet(packet, PAYLOAD_SIZE);
      packet_err = decode_packet(&packet, PAYLOAD_SIZE, 2024, pkt_nb);
      printf("[PROC_BETA] Packet after correction:\n");
      print_packet(packet, PAYLOAD_SIZE);
      if (packet_err == ECC_DE || packet_err == WRONG_SEQ_NB){
         send_ackno_reply(NACK, &con_alfa);
         pkt_nb = pkt_nb;
      } else if (packet_err == ECC_SE){
         send_ackno_reply(ACK_ERR, &con_alfa);
         fputs((char *) packet.payload, output_fptr); // Store the received
packet
         pkt_nb ++;
      } else if (packet_err == NONE){
         send_ackno_reply(ACK, &con_alfa);
         fputs((char *) packet.payload, output_fptr); // Store the received
packet
         pkt_nb ++;
      } else {
         //Do nothing (Different Target)
         pkt_nb ++;
      }
      fclose(output_fptr); //@NOTE: For some reason, we need to open and
close the file for each written packet
   }

   close_connect(&con_alfa);
   close_connect(&con_inject);
   printf("[PROC_BETA] Finishing Process Beta ...\n");
   return 0;
   return 0;
}
```

Injetor e Gerador de Erros
(3_inject_error.c, error_gen.c, error_gen.h)

Geração e injeção de erros (em 7 cenários diferentes) para validação do ECC



Cenário	Descrição
scenario0	Sem Erro
scenario1	Erro Simples aleatoriamente gerado em relação a "flit", "field"e "position".
scenario2	Erro Duplo aleatoriamente gerado em relação a "flit", "field"e "position".
scenario3	Dois Erros Simples em diferentes e aleatoriamente gerados em relação a "flit", "field"e "position".
scenario4	N Erros Simples em diferentes e aleatoriamente gerados em relação a "flit", "field"e "position". Onde N també é aleatório.
scenario5	Erro Simples no campo SH aleatoriamente gerado em relação a "flit", e "position".
scenario6	Erro Simples no campo ECC aleatoriamente gerado em relação a "flit", e "position".

Error Correction Code

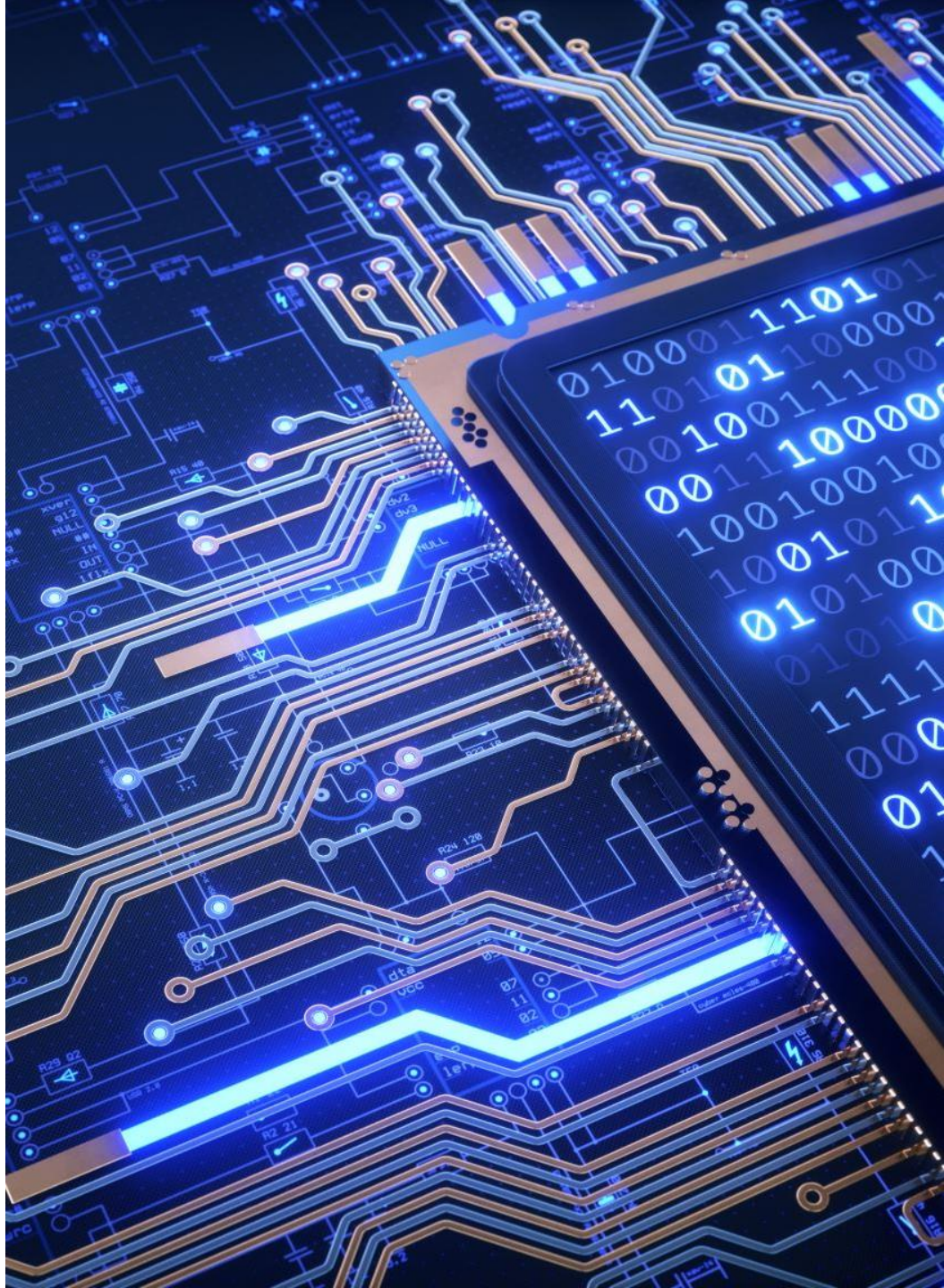
O ECC proposto é o hamming extendido que calcula um número de bits de redundância de modo que seja possível "cobrir"todos os bits da palavra original e, após o cálculo desses bits, um último bit de paridade é calculado executando a operação de ou-exclusivo sobre todos os bits da palavra com os de redundância.

Para que a redundância consiga cobrir toda a palvra, é necessário que a Equação 1.1 seja respeitada, onde r é o núméto de bits de redundância e n é o número de bits da palavra.

$$2^r - 1 \geq n \tag{1.1}$$

Posição do bit		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
bits codificados		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
bits de paridade	p1	x		x		x		x		x		x		x		x		x		x		
	p2		x	x			x	x			x	x			x	x			x	x		...
	p4				x	x	x	x					x	x	x	x					x	
	p8								x	x	x	x	x	x	x	x						
	p16																x	x	x	x	x	

Figura 1.1: Organização dos bits de redundância (p*) na palavra. (Fonte: [Wik20])



Resultados

Apesar de o arquivo ter 100 pacotes, mais pacotes podem ser transmitidos, já que existem situações de retransmissão.

- **Desse modo, ao final da simulação, tivemos 132 pacotes transitados**

Tabela 2.1: Número de ACK, ACK ERROR e NACKS

Estatística	
ACK	40
ACK ERROR	60
NACK	32

Tabela 2.2: Número de execução de cada cenário.

Estatística	
Cenário 0	16
Cenário 1	18
Cenário 2	24
Cenário 3	19
Cenário 4	18
Cenário 5	15
Cenário 6	22

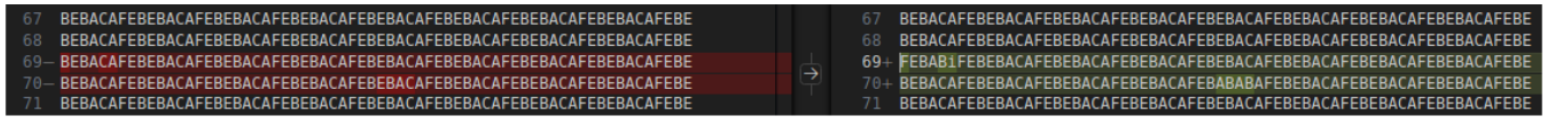


Figura 2.1: Diferença entre os pacotes de entrada e saída.

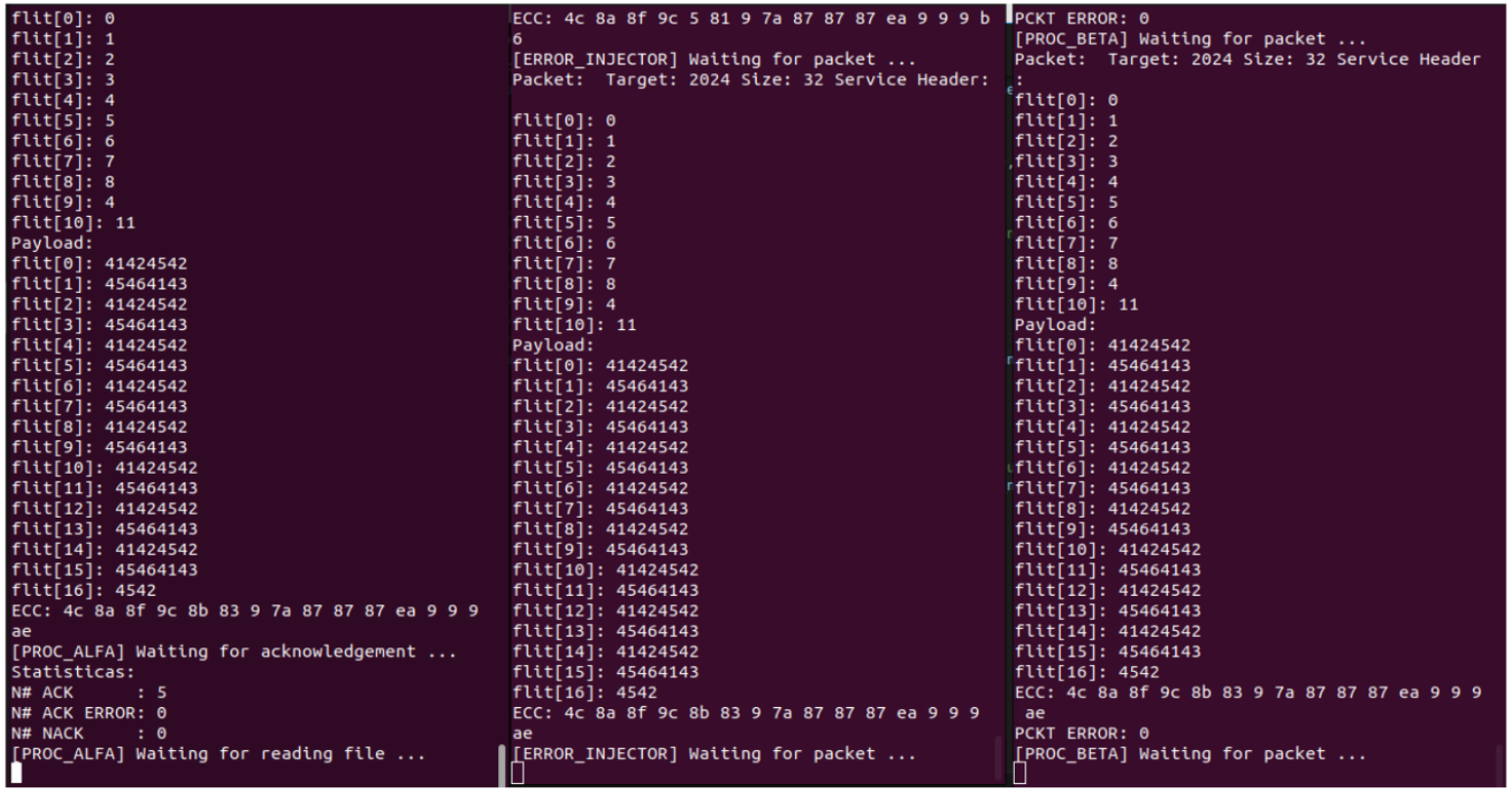


Figura 2.2: Execução dos Processos